

UNIVERSIDAD DE CUENCA

FACULTAD DE INGENIERÍA

ESCUELA DE INFORMÁTICA



**“IMPLEMENTACIÓN GEOSPARQL SOBRE UNA
BASE DE DATOS NOSQL-RDF”**

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL
TÍTULO:
INGENIERO DE SISTEMAS

AUTOR:
PAUL FERNANDO GALLEGOS AGUILAR CI: 0106355431

DIRECTOR: ING. MARCO ANDRÉS TELLO GUERRERO M.Sc. CI:
0704166818
CO-DIRECTOR: ING. VÍCTOR HUGO SAQUICELA GALARZA PhD.
CI: 0103599577

Ecuador, Cuenca - 2017



Resumen

Los datos en formato RDF crecen en gran medida y más aún si se los combina con información geográfica. El estándar GeoSPARQL define todo un vocabulario y funciones para definir información geográfica en RDF. Actualmente, existen herramientas que utilizan diversas tecnologías para almacenar y procesar RDF. Una de ellas son las herramientas NoSQL-RDF que usan tecnologías NoSQL y otras son las que usan bases de relacionales. Las tecnologías NoSQL son optimizadas para el manejo de grandes volúmenes de información y procesamiento distribuido, mientras que las relacionales no lo son. Dentro de las herramientas que usan bases de datos relacionales existe una implementación GeoSPARQL, sin embargo para herramientas NoSQL-RDF aún no existe.

En este trabajo se implementó el estándar GeoSPARQL sobre una base de datos NoSQL-RDF para así aprovechar las ventajas de las tecnologías NoSQL sobre grandes volúmenes de datos RDF. La implementación está formada por dos componentes, el procesador de consultas y el procesador de operaciones que trabajan conjuntamente para procesar GeoSPARQL. Una vez finalizada la implementación se evaluó su funcionamiento mediante la ejecución y comparación de los resultados obtenidos con los resultados que proporciona la implementación GeoSPARQL realizada sobre Apache Marmotta.

Palabras clave: GeoSPARQL, Web Semántica, RDF, NoSQL, NoSQL-RDF, geografía, geoespacial, filtro, topología, procesador, componente.



Abstract

Data in RDF format grows greatly and more so if combined with geographic information. The GeoSPARQL standard defines a vocabulary and functions for defining geographic information in RDF. Currently, there are tools that use various technologies to store and process RDF. One of them are the NoSQL-RDF tools that use NoSQL technologies and others are those that use relational databases. NoSQL technologies are optimized for handling large volumes of information and distributed processing, while relational ones are not. Within tools that use relational databases there is a GeoSPARQL implementation, however for NoSQL-RDF tools does not yet exist.

In this work, the GeoSPARQL standard was implemented on a NoSQL-RDF database to take advantage of the NoSQL technologies on large volumes of RDF data. The implementation consists of two components, the query processor and the operations processor that work together to process GeoSPARQL. Once the implementation was completed, its operation was evaluated by the execution and comparison of the results obtained from our tool with those provided by the GeoSPARQL implementation performed on Apache Marmotta.

Keywords: GeoSPARQL, Semantic web, RDF, NoSQL, NoSQL-RDF, geography, geospatial, filter, topology, processor, component.



Índice general

Resumen	1
Abstract	2
Dedicatoria	10
Agradecimientos	11
1. Introducción	12
1.1. Identificación del problema	12
1.2. Justificación	13
1.3. Alcance	13
1.4. Objetivo General	14
1.5. Objetivos Específicos	14
2. Marco Teórico	15
2.1. Web Semántica	15
2.1.1. Definición	15
2.2. Tecnologías para la web semántica	15
2.2.1. RDF	16
2.2.2. SPARQL	16
2.2.3. GeoSPARQL	17
2.3. Base de datos NoSQL	19
2.3.1. Definición	21
2.3.2. Finalidad de las bases de datos NoSQL	21
2.3.3. ¿Quiénes usan bases de datos NoSQL?	21
2.4. Herramientas NoSQL-RDF	22
2.4.1. Definición	22
2.4.2. Tipos	22
2.4.3. Herramientas NoSQL-RDF analizadas	24
2.5. Herramientas que soportan operaciones geoespaciales	28
2.5.1. GeoTools	28
2.5.2. PostGIS	30



3. Diseño e implementación de la extensión GeoSPARQL sobre CumulusRDF	32
3.1. Diseño de la arquitectura	32
3.1.1. Procesador de consultas GeoSPARQL	32
3.1.2. Procesador de operaciones geográficas	35
3.2. Diseño e implementación del procesador de consultas	36
3.3. Diseño e implementación del procesador de operaciones	39
3.4. Integración GeoSPARQL con CumulusRDF	42
4. Evaluación de resultados	43
4.1. Equipo	43
4.2. Datos de prueba	43
4.3. Ejecución de las consultas de pruebas	44
Conclusiones	47
Apéndices	48
A. Instalación Single-Node de Apache Cassandra	49
B. Instalación de CumulusRDF	51
C. Configuración de CumulusRDF para implementación de la extensión GeoSPARQL	54
D. Configuración del clúster Multi-Node de Apache Cassandra	57
E. Definición de carga y consulta de datos GeoSPARQL para CumulusRDF	60



Índice de figuras

2.1. La web actual vs. la web semántica [5]	16
2.2. Grafo RDF	17
2.3. Dominio de la ontología GeoSPARQL [2]	18
2.4. Empresas que usan bases de datos NoSQL	22
2.5. Taxonomía de los tipos de almacenamiento y esquemas de partición utilizados por las herramientas NoSQL-RDF [12].	23
2.6. Estructura de CumulusRDF	25
2.7. Estructura de PigSPARQL	27
2.8. Estructura de GeoTools	29
2.9. Estructura de PostGIS	30
3.1. Arquitectura de la extensión GeoSPARQL con CumulusRDF	33
3.2. Ilustración del procesador de consultas	33
3.3. Ilustración del procesador de operaciones	37
3.4. Diagrama de clases del procesador de consultas	38
3.5. Diagrama de clases de las relaciones topológicas binarias Simple Features	40
3.6. Diagrama de clases de las funciones filtro geoespaciales	41
3.7. Diagrama que une las relaciones topológicas binarias Simple Features y las ficciones filtro geoespaciales.	41
3.8. Diagrama que une la integración de los procesadores a CumulusRDF	42
4.1. a) Resultado de la implementación GeoSPARQL sobre CumulusRDF. b) Resultado de la implementación GeoSPARQL sobre Apache Marmotta.	45
4.2. a) Resultado de la implementación GeoSPARQL sobre CumulusRDF. b) Resultado de la implementación GeoSPARQL sobre Apache Marmotta.	46
A.1. Consola de Cassandra	50
B.1. Estructura general del proyecto	52
B.2. Perfil seleccionado para la compilación	52
B.3. Orden de compilación de módulos	52
B.4. Módulos compilados	53
B.5. Página de inicio de CumulusRDF	53



C.1. Modificación del archivo pom.xml	54
C.2. Estructura del proyecto CumulusRDF	55
C.3. Dependencia Jena	55
C.4. Dependencia GeoTools	56
C.5. Repositorio GeoTools	56
D.1. Consola de Apache Cassandra	59
D.2. Estado del cluster	59
E.1. Configuración Single-Node	61
E.2. Configuración Multi-Node	61



Índice de tablas

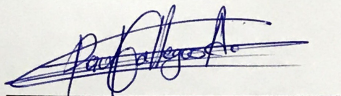
2.1. Relaciones topológicas binarias Simple Features	19
2.2. Filtros Geoespaciales	20
2.3. Índices de almacenamiento de CumulusRDF; S (Subject), P (Predicate), O (Object), C (Context).	26
2.4. Patrones triples con sus tablas índice de consulta	28
3.1. Condiciones geográficas	35
3.2. Lista de resultados SPARQL	36
E.1. Opciones para la carga de datos	62
E.2. Opciones para la consulta GeoSPARQL	62

Cláusula de licencia y autorización para publicación en el Repositorio
Institucional

PAUL FERNANDO GALLEGOS AGUILAR en calidad de autor y titular de los derechos morales y patrimoniales del trabajo de titulación "IMPLEMENTACIÓN GEOSPARQL SOBRE UNA BASE DE DATOS NOSQL-RDF", de conformidad con el Art. 114 del CÓDIGO ORGÁNICO DE LA ECONOMÍA SOCIAL DE LOS CONOCIMIENTOS, CREATIVIDAD E INNOVACIÓN reconozco a favor de la Universidad de Cuenca una licencia gratuita, intransferible y no exclusiva para el uso no comercial de la obra, con fines estrictamente académicos.

Asimismo, autorizo a la Universidad de Cuenca para que realice la publicación de este trabajo de titulación en el repositorio institucional, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación Superior.

Cuenca, octubre de 2017



Paul Fernando Gallegos Aguilar

C.I: 0106355431



Cláusula de Propiedad Intelectual

PAUL FERNANDO GALLEGOS AGUILAR, autor del trabajo de titulación "IMPLEMENTACIÓN GEOSPARQL SOBRE UNA BASE DE DATOS NOSQL-RDF", certifico que todas las ideas, opiniones y contenidos expuestos en la presente investigación son de exclusiva responsabilidad de su autor.

Cuenca, octubre de 2017

Paul Fernando Gallegos Aguilar

C.I: 0106355431



Dedicatoria

*La presente tesis la dedico de manera especial
a mis padres Alejandro y Virginia, que me dieron
su completa confianza, apoyo moral, he hicieron
la persona que he llegado a ser.*



Agradecimientos

*En primer lugar quiero agradecer a mis padres,
por todo el sacrificio que hicieron al brindarme
la oportunidad de estudiar una carrera profesional,
por impulsarme con sus palabras de aliento y por último,
ser el mejor ejemplo de dedicación y perseverancia.*

*También quiero agradecer a mi institución,
a mis maestros y sobre todo a mis directores de
tesis, quienes dieron su tiempo y proporcionaron su experiencia
para llegar a terminarla de manera exitosa.*



Capítulo 1

Introducción

1.1. Identificación del problema

El nacimiento del internet marcó la nueva era de la información, y en sus inicios su contenido fue hecho para el uso humano, a pesar de que todo es legible y comprensible, esto no lo es para una máquina [20]. Día tras día el volumen de información crece a grandes escalas, y cada vez es más compleja tratar de manipularla y prácticamente es imposible hacerlo manualmente. Una de las soluciones es el uso de metadatos, los cuales describen los datos contenidos en ellos mediante empaquetadores genéricos permitiendo así la integración y comunicación entre distintos esquemas de meta-datos [8]. RDF (Resource Description Framework) tiene la finalidad de crear una infraestructura para la descripción de recursos y este a su vez crea la interoperabilidad entre aplicaciones que intercambian información en la web [1]. La utilización de RDF mejora las prestaciones de motores de búsqueda, software inteligente (bots), descripción de páginas web, entre otros [14].

Una de las áreas en que la informática ha puesto énfasis, es la información geográfica, por los crecientes datos que se obtiene a través de sensores, satélites y otros dispositivos alrededor del mundo. Los metadatos geoespaciales describen recursos geográficos, tomando la información geográfica para modelarlos como líneas, puntos o polígonos. Usando dicho modelo se puede describir la información de lugares, países, ciudades, ríos, etc [19]. Actualmente, existen herramientas que permiten explotar este tipo de datos mediante operaciones geoespaciales (INSERTS, WITHIN, CONTAINS, OVERLAPS, etc.) para así obtener información de lugares, rutas óptimas para transporte, tendencias de situaciones temporales, etc. [9, 17].

Existen herramientas que hacen uso de bases de datos relacionales para almacenar y procesar los metadatos, pero, estos tienen lentitud de respuesta cuando procesan grandes volúmenes de información, y más aún con la información geoespacial que crece día tras día. Para hacer frente a ese problema, el nacimiento de las herramientas NoSQL-RDF se justifica, proveyendo las bondades y características de las bases de datos NoSQL al proce-



so y almacenamiento de metadatos, sin embargo estas aun no soportan el procesamiento de operaciones geoespaciales.

1.2. Justificación

A pesar de que ya se haya sustentado el problema de manejo de grandes volúmenes de información con el uso de bases de datos NoSQL, la inexistencia de un soporte GeoSPARQL sobre una herramienta NoSQL-RDF motiva la implementación del mismo, para así aprovechar las ventajas que proporcionan las bases de datos NoSQL. Manejo de grandes volúmenes de información, mejores tiempos de respuesta sobre datos geoespaciales motivan el logro de este objetivo.

1.3. Alcance

Se trabajará con información geoespacial en formato RDF que posteriormente será almacenada en una base de datos NoSQL a través de una herramienta NoSQL-RDF. La mejora radica únicamente en el reconocimiento y procesamiento de consultas GeoSPARQL por parte de una herramienta NoSQL-RDF a través de la extensión a implementar. Finalmente, una vez obtenido la implementación GeoSPARQL se probará su funcionamiento sobre un clúster.



1.4. Objetivo General

Implementar una extensión que de soporte GeoSPARQL sobre una base de datos NoSQL-RDF

1.5. Objetivos Específicos

1. Analizar información geoespacial.
2. Analizar herramientas que den soporte a operaciones geoespaciales.
3. Analizar herramientas NoSQL-RDF.
4. Diseñar un modelo de almacenamiento RDF geoespacial para una base de datos NoSQL.
5. Diseñar un modelo de consulta GeoSPARQL para una base de datos NoSQL-RDF.
6. Implementar el modelo de almacenamiento y consulta para una base de datos NoSQL-RDF.
7. Realizar pruebas de consultas GeoSPARQL.
8. Realizar pruebas de consultas GeoSPARQL en un clúster de servidores.



Capítulo 2

Marco Teórico

En este capítulo se presenta toda la base teórica en la cual se sostiene este trabajo, se describirán conceptos, herramientas y terminologías.

2.1. Web Semántica

La Web actual, está constituida por páginas enlazadas, básicamente describe la forma de mostrar la información, pero no su significado. Esto implica que las búsquedas se realizan a través de coincidencias de palabras clave sin ninguna relación semántica con los contenidos de las páginas. La Web semántica, como una nueva visión, propone añadir un componente descriptivo a los recursos disponibles, dando relaciones y significado a su contenido [4]. En la figura 2.1 se puede observar la gran diferencia entre la web actual (páginas enlazadas) y la web semántica (contenido definido y relacionado).

2.1.1. Definición

La web semántica es una extensión de la web actual, en la que a la información disponible se le otorga un significado bien definido, permitiendo a los ordenadores y personas trabajar en cooperación. Está basada en la idea de proporcionar en la Web datos definidos y enlazados, haciendo que las aplicaciones heterogéneas localicen, integren, razonen y reutilicen la información presente en la Web [3].

2.2. Tecnologías para la web semántica

El desarrollo de la web semántica también originó nuevas tecnologías, lenguajes semánticos, y otros. El primer lenguaje para la construcción de la web semántica fue SHOE, creado por Jim Hendler en la Universidad de Maryland en 1997 y desde entonces varios lenguajes y estándares se han originado con la misma finalidad [5].

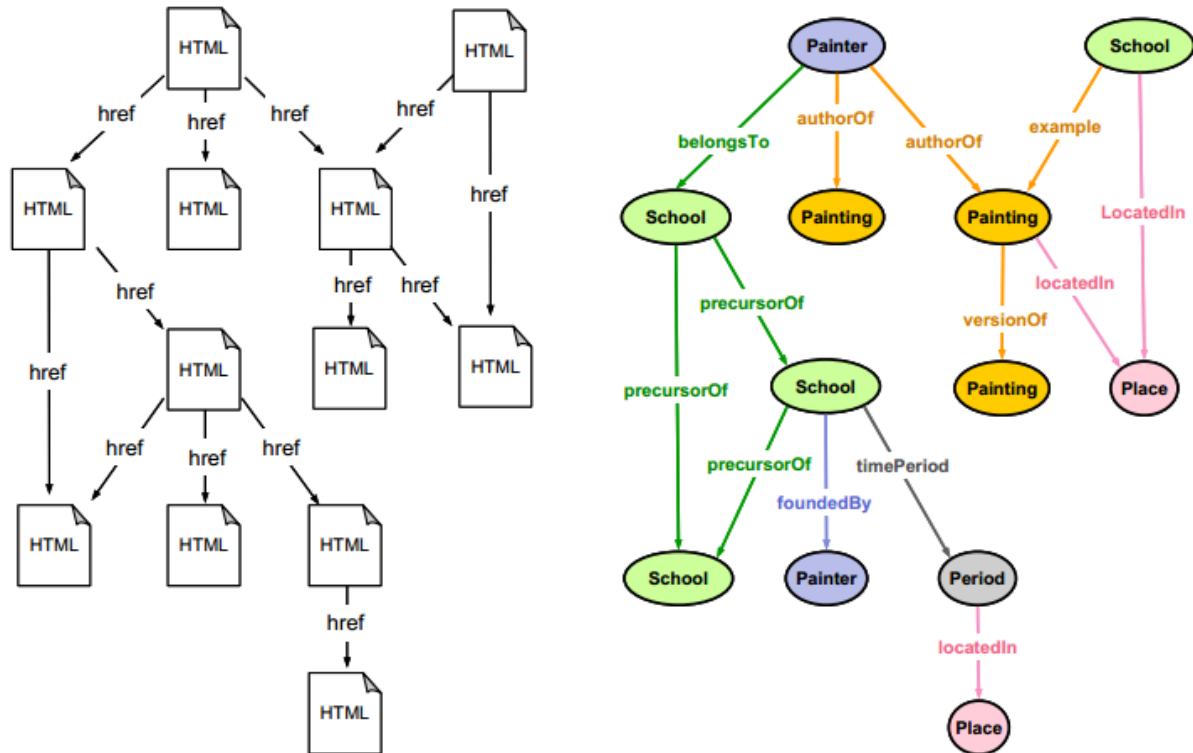


Figura 2.1: La web actual vs. la web semántica [5]

2.2.1. RDF

RDF¹ es un modelo de datos donde reside la estructura semántica no ambigua de los recursos de la web de distinto dominio (medicina, ingeniería, electrónica, geografía etc.). Está basado en la identificación de recursos mediante Uniform Resource Identifiers o URIs [24]. La descripción en RDF se resume en preposiciones simples compuesta por sujeto, predicado y objeto (s, p, o), conocido como tripleta [12]. Cualquier preposición se puede representar claramente utilizando una tripleta, pero, para representar en RDF se utiliza la notación de grafo dirigido, tomando a sus nodos para representar sujeto y objeto, y sus arcos como predicado (Ver Figura 2.2).

2.2.2. SPARQL

SPARQL² es un lenguaje estandarizado para hacer consultas sobre grafos RDF, de la misma manera que lo hace SQL³ sobre una base de datos relacional. Las consultas SPARQL trabajan mediante la búsqueda de patrones que coincidan dentro de un grafo

¹Resource Description Framework

²Protocol and RDF Query Language

³Structured Query Language

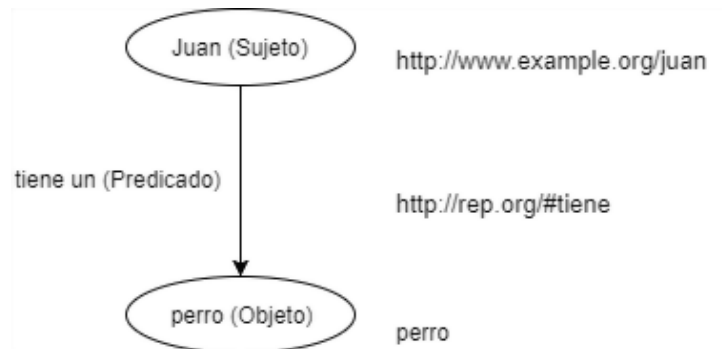


Figura 2.2: Grafo RDF

RDF y se representa mediante un conjunto de variables y condiciones parametrizadas. Los resultados de una consulta será un grafo reducido sujeto a las condiciones declaradas [25].

2.2.3. GeoSPARQL

A diferencia de SPARQL que es un protocolo y lenguaje de consulta para RDF, GeoSPARQL es un lenguaje de consulta para datos geográficos en RDF. Define una extensión de vocabulario para representar datos espaciales en RDF y otra a SPARQL para consultar dichos datos [15].

Sus características están definidas por:

- Un vocabulario RDF / OWL⁴ para representar información espacial.
- Un conjunto de funciones para cálculos espaciales.
- Un Conjunto de reglas de transformación de consultas.

Ontología GeoSPARQL

La ontología GeoSPARQL, para representar características y geometrías incluye la clase geo: SpatialObject, con dos subclases principales, geo: Feature y geo: Geometry [2].

En la figura 2.3 se muestra la constitución del dominio de la ontología, en ella existen clases que se instancian (geo:Feature), y características que se relacionan a través de sus propiedades (geo: hasGeometry). GeoSPARQL proporciona diferentes clases para las jerarquías geométricas asociadas a las representaciones de muchos tipos de diferentes geometrías, como punto, polígono, curva, arco y multicurva. También incluye dos formas

⁴Web Ontology Language

diferentes de representar literales geométricos, asociados a los tipos WKT⁵ y GML⁶. Las propiedades geo: asWKT y geo: asGML vinculan las entidades geométricas a las representaciones literales geométricas [2, 15].

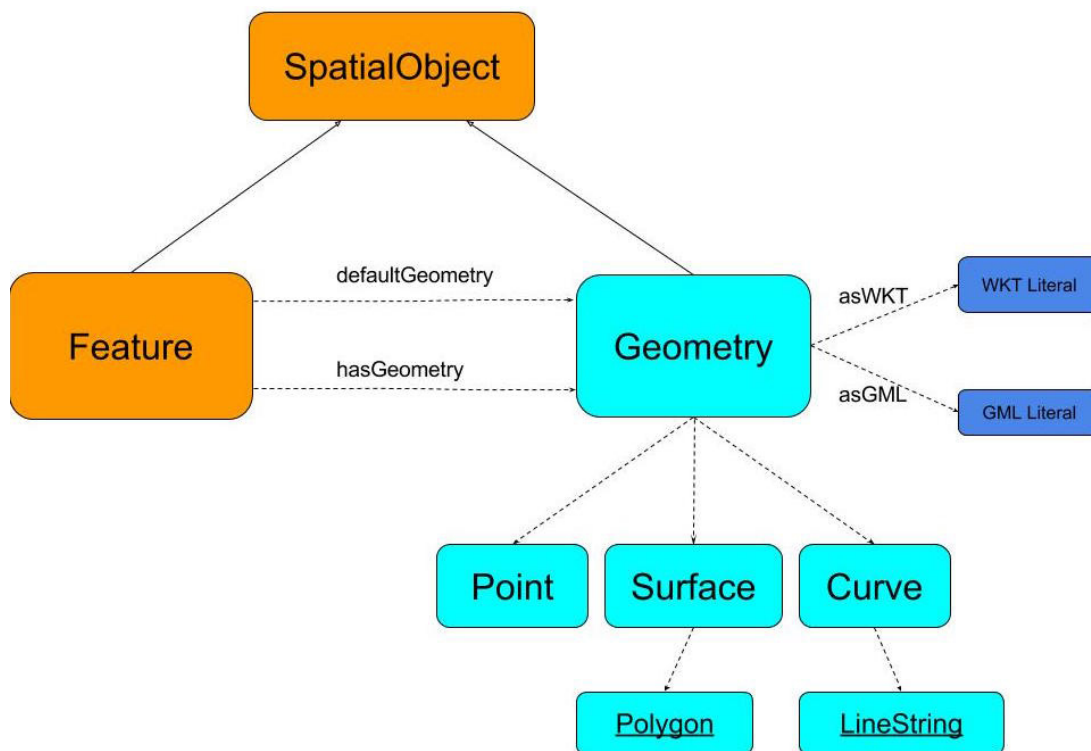


Figura 2.3: Dominio de la ontología GeoSPARQL [2]

Relaciones espaciales

GeoSPARQL incluye formas estándar para hacer consultas, estas son:

- Relaciones topológicas binarias.
- Funciones filtro geoespaciales.

Las relaciones topológicas binarias pueden usarse en patrones triples de consulta. Principalmente se utilizan entre objetos del tipo geo: Geometry. Se pueden expresar utilizando tres vocabularios distintos, estos son: The OGC⁷ Simple Features, Egenhofer 9-intersection model y RCC8. En la tabla 2.1 se muestra la lista de las relaciones topológicas binarias.

⁵Well know Text

⁶Geography Markup Language

⁷Open Geospatial Consortium

Función	Sintaxis	Descripción
equals	geof:sfEquals (geom1, geom2)	Devuelve verdadero si dos geometrías topológicamente son iguales.
disjoint	geof:sfDisjoint (geom1, geom2)	Devuelve verdadero si dos geometrías están disjuntas.
intersects	geof:sfIntersects (geom1, geom2)	Devuelve verdadero si dos geometrías se intersectan.
touches	geof:sfTouches (geom1, geom2)	Devuelve verdadero si las dos geometrías se tocan; y falso si ambas Geometrías son puntos.
within	geof:sfWithin (geomA, geomB)	Devuelve verdadero si la geometría A esta dentro de la geometría B.
contains	geof:sfContains (geomA, geomB)	Devuelve verdadero si la geometría A contiene a la geometría B.
overlaps	geof:sfOverlaps (geom1, geom2)	Devuelve verdadero si dos geometrías se superponen.
crosses	geof:sfCrosses (geom1, geom2)	Devuelve verdadero si dos geometrías se cruzan.

Tabla 2.1: Relaciones topológicas binarias Simple Features

Las funciones de filtro geoespaciales proporcionan dos tipos diferentes de funcionalidad. La primera tiene funciones que toman múltiples geometrías como predicados y producen una nueva geometría u otro tipo de datos como resultado y la segunda son relaciones topológicas binarias [2].

Como se puede observar existen dos tipos de relaciones topológicas binarias; las del primer grupo dichas anteriormente y las que pertenecen a las funciones filtro. La principal diferencia radica en que la primera toma los literales geométricos como parámetros, mientras que las otras toman a las entidades geo:Geometry y geo:Feature (Ver Tabla 2.2).

2.3. Base de datos NoSQL

La creciente información en la red hace que los motores tradicionales de procesamiento y consulta tengan grandes problemas de escalabilidad y rendimiento ante las millones de consultas que deben procesar sus bases de datos relacionales. El objetivo de las grandes compañías de internet es presentar resultados de forma rápida y eficiente entre los grandes volúmenes de información [21]. Con dicha problemática surge el concepto las bases de datos NoSQL.

Función	Sintaxis	Característica
geof:distance	geof:distance (geom1, geom2, units)	Devuelve la distancia más corta entre dos puntos de dos objetos geométricos.
geof:buffer	geof:buffer (geom1, radius, units)	Esta función devuelve un objeto geométrico que representa todos los puntos cuya distancia desde geom1 es menor o igual que el radio medido en unidades.
geof:convexHull	geof:convexHull (geom1)	Esta función devuelve un objeto geométrico que representa todos los puntos en el casco convexo de geom1.
geof:intersection	geof:intersection (geom1,geom2)	Esta función devuelve un objeto geométrico que representa la intersección de todos los puntos de geom1 con geom2.
geof:union	geof:union (geom1, geom2)	Esta función devuelve un objeto geométrico que representa la unión de todos los puntos de geom1 con geom2.
geof:difference	geof:difference (geom1, geom2)	Esta función devuelve un objeto geométrico que representa la diferencia del conjunto puntos de geom1 con geom2.
geof:symDifference	geof:symDifference (geom1, geom2)	Esta función devuelve un objeto geométrico que representa la diferencia simétrica de todos los puntos de geom1 con geom2.
geof:boundary	geof:boundary (geomA)	Esta función devuelve el cierre del límite de geom1.
geof:getsrid	geof:getSRID (geom)	Devuelve el URI del sistema de referencia espacial para geom.
geof:envelope	geof:envelope (geom1)	Esta función devuelve el cuadro delimitador mínimo de geom1.

Tabla 2.2: Filtros Geoespaciales



2.3.1. Definición

NoSQL son sistemas de almacenamiento de información que no cumplen el esquema entidad-relación es decir no impone la relación de tablas ni relaciones entre ellas, además su información puede ser almacenada en distintos formatos, clave-valor, documentos, columnas o grafos [13, 21].

2.3.2. Finalidad de las bases de datos NoSQL

Las bases de datos NoSQL surgen por el cambio de manejo de la información por parte de las empresas, ya que no solo desean almacenar esta información, sino quieren sacarle el mayor provecho; además los usuarios piden cada vez más velocidad en las consultas. Además, la información crece a un ritmo sin precedentes, y cada vez se hace más compleja su administración [21].

Características Generales:

Las bases de datos NoSQL poseen varias características, y cada una puede tener ventajas o desventajas según ámbito en el cual se utilice [21]. A continuación se nombran las características comunes que ellas poseen:

1. **Libertad de esquema:** Permite mayor libertad para modelar los datos.
2. **Estructura distribuida:** Habilidad de replicar y distribuir los datos sobre los servidores que geográficamente pueden estar en distintos lugares.
3. **Escalabilidad horizontal:** Puede añadir o eliminar elementos de hardware al sistema, sin afectar el rendimiento.
4. **Consultas simples:** Al no tener relaciones, evita hacer operaciones de JOIN u otras operaciones, simplificando así sus consultas.
5. No Implementa o implementa parcialmente transacciones ACID⁸

2.3.3. ¿Quiénes usan bases de datos NoSQL?

En la figura 2.4 se puede observar a muchas empresas importantes que han decidido utilizar este tipo de bases de datos, llegando a potenciar su desempeño y eficacia frente a los grandes volúmenes de información que generan.

⁸Atomicity, Consistency, Isolation and Durability



Figura 2.4: Empresas que usan bases de datos NoSQL

2.4. Herramientas NoSQL-RDF

En la actualidad existen una serie de herramientas que son capaces de almacenar RDF en una base de datos NoSQL y consultarlos a través de SPARQL. Muchas de ellas continúan vigentes y están en continuo desarrollo, sin embargo otras quedaron como proyectos olvidados o son software propietario. La gran variedad de las mismas viene de la mano de la diversidad de tecnologías de bases de datos NoSQL existentes en el mercado.

2.4.1. Definición

A este tipo de herramientas se puede definir como repositorios que se centran en almacenar y administrar grandes cantidades de información RDF a través de tecnologías NoSQL [11].

2.4.2. Tipos

Los sistemas de archivos distribuidos, bases de datos clave/valor, almacenamiento centralizado y mezclas entre ellas son ampliamente utilizadas por las herramientas NoSQL-RDF para almacenar y administrar sus datos. En la figura 2.5 se puede observar una clasificación acorde al tipo de base de datos que utilizan, los cuales se describen a continuación:

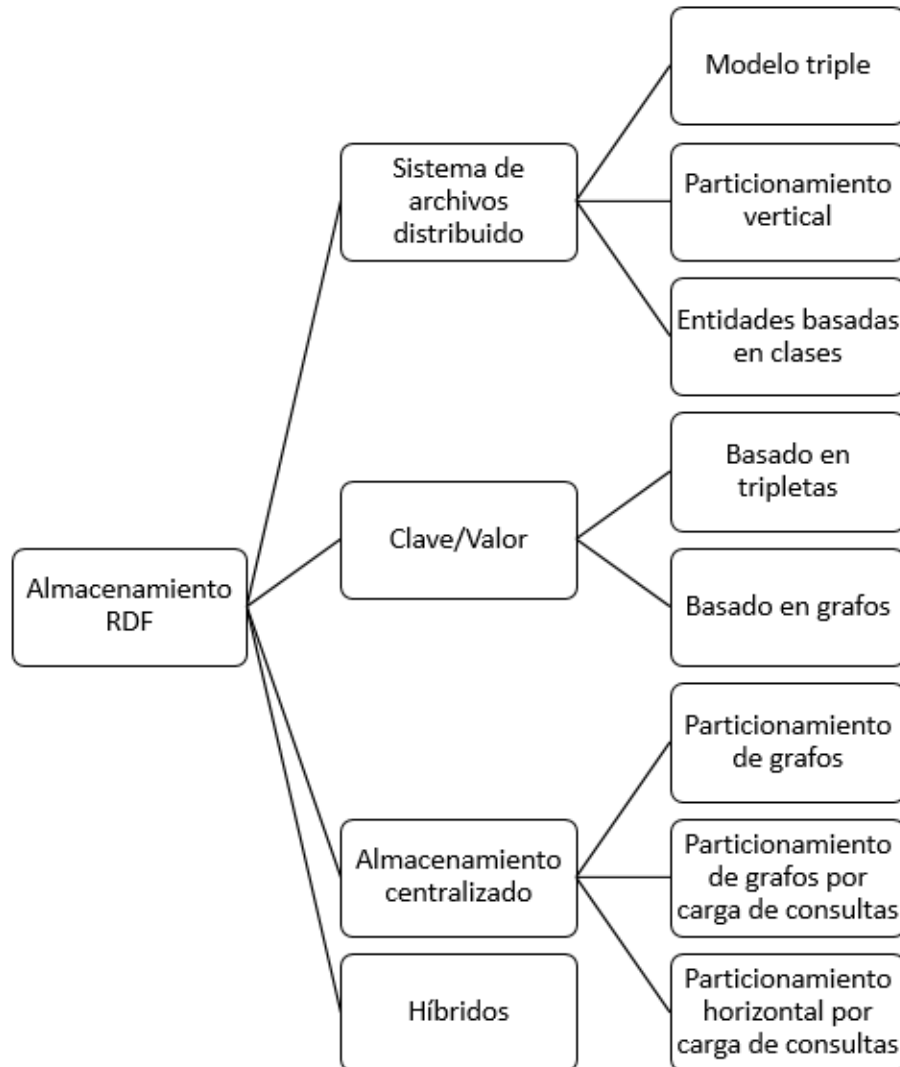


Figura 2.5: Taxonomía de los tipos de almacenamiento y esquemas de partición utilizados por las herramientas NoSQL-RDF [12].

Sistema de archivos distribuidos

Están diseñados para almacenamiento confiable de datos en un grupo de máquinas. Los archivos grandes se dividen en pequeños fragmentos que son distribuidos entre las máquinas del clúster y replicadas automáticamente por el servidor maestro del DFS⁹. Esto hace que sea tolerante a fallos. El servidor maestro del DFS suele ser el nodo responsable de replicar y hacer un seguimiento de los diferentes fragmentos de datos que componen cada archivo. Un gran defecto de los sistemas de archivos distribuidos es que no proporcionan

⁹Distributed file systems

acceso de datos de gran precisión y, por lo tanto, el acceso a una parte de los datos sólo puede lograrse mediante un análisis completo de todos los archivos [12]. Los sistemas que utilizan DFS para almacenar RDF se clasificaron de la siguiente manera:

- **Modelo triple:** Almacena la estructura común RDF.
- **Particionamiento vertical:** Para almacenar RDF divide el archivo de tripletas RDF en archivos más pequeños.
- **Entidades basadas en clases:** Utiliza grafos de alto nivel para el particionamiento de tripletas RDF.

Clave/valor

Esta importante familia de bases de datos NoSQL provee una estructura simple, basada en el concepto (clave, valor). Proporciona almacenamiento eficiente de gran precisión y recuperación de datos, adecuados para la pequeña granularidad de RDF [12]. Los sistemas que usan esta tecnología se ha clasificado de la siguiente manera:

- **Basado en tripletas:** Usan un sistema de indexación.
- **Basado en grafos:** A nivel de almacenamiento considera a los grafos RDF de forma holística.

Almacenamiento centralizado

Esta categoría comprende a los sistemas que explotan en paralelo un conjunto de almacenes RDF centralizados distribuidos entre muchos nodos. Estos sistemas tienen una arquitectura maestro/esclavo, donde el maestro es responsable de particionar y colocar las tripletas RDF en los nodos esclavos [12].

Híbridos

Son sistemas que explotan una combinación de un sistema de archivos distribuido, clave-valor y almacenamiento centralizado [12].

2.4.3. Herramientas NoSQL-RDF analizadas

Las diferentes formas y diseños para almacenar información RDF en las bases de datos NoSQL por parte de las herramientas NoSQL-RDF hacen que sea difícil seleccionar alguna y definirla como la mejor. Cada una de las herramientas tendrá mejor o peor desempeño acorde a las circunstancias para las que se las utiliza. Por ejemplo, una herramienta que utilice una base de datos clave/valor será más eficiente al momento de encontrar un solo valor de entre un conjunto de datos, mientras que una que utilice un sistema de

archivos distribuido tendría que analizar toda su información, una a una para encontrar ese mismo valor, ralentizando así su desempeño, ya que esta es más apropiada para analizar información por lotes. Muchas de ellas continúan en desarrollo y otras han quedado en el olvido. A continuación se muestran algunas herramientas que fueron objeto de análisis para este trabajo.

CumulusRDF

CumulusRDF es un almacén RDF escrito en Java que proporciona búsquedas en tripletas, carga masiva y consulta a través de SPARQL. El sistema de almacenamiento de CumulusRDF es Apache Cassandra, el cual fue desarrollado por Facebook. Apache Cassandra proporciona almacenamiento de datos descentralizado y tolerancia a fallos basados en replicación y failover¹⁰ [6].

CumulusRDF está compuesto por SesameSail¹¹ y Apache Cassandra, que son utilizados para analizar, almacenar y hacer consultas SPARQL sobre la información RDF, en la figura 2.6 se muestra su estructura [7].

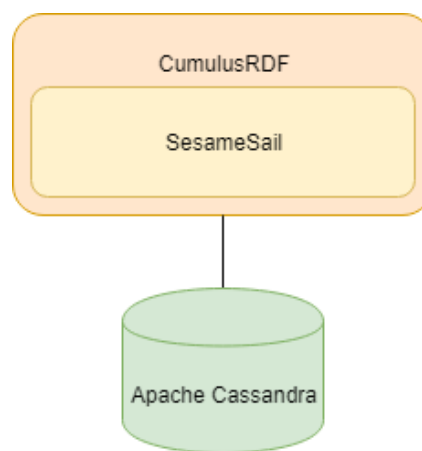


Figura 2.6: Estructura de CumulusRDF

Tanto las consultas SPARQL como la información RDF son procesadas por SesameSail dentro de CumulusRDF. Cada tripleta RDF es procesada y almacenada en el esquema de datos alojado en Apache Cassandra que está conformado por los diferentes índices (Ver Tabla 2.3). El procesador de consultas de SesameSail traduce SPARQL a CQL¹² para consultar los datos almacenados en los índices alojados en Apache Cassandra. CumulusRDF

¹⁰Es un modo de funcionamiento, en el que si falla un componente en el sistema, sus funciones son asumidos por otro de sus componentes.

¹¹Framework que procesa consultas y datos RDF

¹²Lenguaje de consultas de Apache Cassandra

Índice	Tipo de almacenamiento
SPOC	Tripletas
OSPC	Tripletas
POSC	Tripletas
OCPS	Quads
SCOP	Quads
SPCO	Quads

Tabla 2.3: Índices de almacenamiento de CumulusRDF; S (Subject), P (Predicate), O (Object), C (Context).

verifica la forma y tipo consulta SPARQL ingresada y tomará el índice apropiado para consultar [6].

Características

- Almacenamiento RDF altamente escalable para aplicaciones de escritura intensa.
- Actúa como servidor de datos enlazados
- Permite una evaluación rápida y ligera de consultas de tripletas
- Tiene total soporte para almacenamiento de tripletas y quads
- Soporta SPARQL 1.1

PigSPARQL

PigSPARQL es una herramienta destinada al procesamiento de consultas SPARQL. Su sistema de almacenamiento alojado en Hadoop ofrece la posibilidad de almacenar y analizar gran cantidad de datos no estructurados, además de soportar una ejecución de manera distribuida. PigSPARQL traduce SPARQL 1.0 a Pig Latin, que permite ejecutar consultas SPARQL en grandes grafos RDF con MapReduce, es decir, utiliza Pig Latin¹³ como una capa intermedia de abstracción entre SPARQL y MapReduce¹⁴[23] (Ver Figura 2.7).

¹³Es un lenguaje para el análisis para grandes conjuntos de datos desarrollados por Yahoo!, basado en el Framework Apache Hadoop

¹⁴MapReduce es un framework que proporciona un sistema de procesamiento de datos paralelo y distribuido

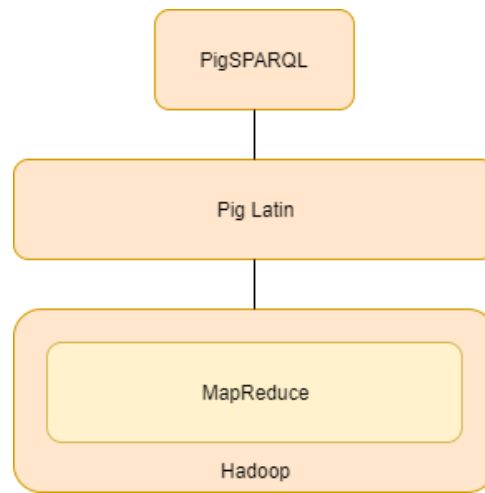


Figura 2.7: Estructura de PigSPARQL

Características

- Particularmente es adecuado para el procesamiento de consultas ad-hoc.
- En una típica consulta SPARQL, el predicado de un patrón triple suele estar limitado.
- Admite el particionamiento vertical opcional del conjunto de datos como un paso adicional de preprocesamiento.

Rya

Rya es un sistema de almacenamiento de datos RDF basado en la nube que soporta las consultas SPARQL. Rya es capaz de realizar el procesamiento de consultas que escalan a billones de tripletas a través de múltiples nodos distribuidos [22]. Rya implementa un algoritmo de bucles anidados de índices utilizando múltiples búsquedas de valores clave [12]. La gestión de los datos RDF están contruidos sobre Apache Accumulo el cual es una base de datos clave/valor [22].

Almacenamiento

El almacenamiento de datos de Rya está compuesto por tablas índices alojadas en Accumulo, los mismo que son consultados por patrones que apuntan a dichas tablas (Ver Tabla 2.4). Accumulo ordena y divide todos los pares clave/valor basados en el ID de la fila clave. Esto significa que a medida que los datos crecen, las filas se agrupan y se ordenan, proporcionando acceso de lectura y escritura muy rápido a rangos cortos en un conjunto de datos grande [18].

Patrones triple	Tablas índice en Accumulo
(subject, predicate, object)	SPO
(subject, predicate, *)	SPO
(subject, *, object)	OSP
(*, predicate, object)	POS
(subject, *, *)	SPO
(*, predicate, *)	POS
(*, *, object)	OSP
(*, *, *)	SPO

Tabla 2.4: Patrones triples con sus tablas índice de consulta

Selección de herramienta

Dentro la investigación y análisis de CumulusRDF, PigSPARQL y Rya, también se realizaron pruebas de funcionamiento. PigSPARQL es una herramienta construida en java, es liviana y de fácil instalación, sin embargo no posee una documentación de su funcionamiento y de su código, además de ser una herramienta anticuada ya que utiliza librerías compatibles con la versión 1.6 de java. Rya actualmente está alojada en una incubadora de apache en un constante desarrollo, su descarga e instalación es bastante compleja y al momento de probarla hubieron muchos problemas de compilación de sus fuentes. Por último, CumulusRDF posee una enriquecida documentación tanto del código como de su funcionamiento, además de ser de fácil instalación. Como conclusión, CumulusRDF fue elegida para la implementación de la extensión GeoSPARQL.

2.5. Herramientas que soportan operaciones geoespaciales

La mayoría de las personas están familiarizadas con el uso de mapas para referirse a ubicaciones físicas en el planeta. Dentro de la informática, la información sobre ubicaciones físicas son tomadas como datos geoespaciales que son representados a través de líneas, puntos o polígonos, además de existir herramientas capaces de procesar dichos datos.

A continuación se mostrará las herramientas que fueron objeto de análisis para este proyecto:

2.5.1. GeoTools

GeoTools es una biblioteca de código abierto LGPL¹⁵ escrito en Java que proporciona métodos compatibles con los estándares para la manipulación de datos geoespaciales.

¹⁵GNU Lesser General Public License

La biblioteca de GeoTools implementa las especificaciones de la OGC¹⁶ a medida que se desarrollan [9].

En la figura 2.8 se puede observar la estructura interna de GeoTools donde cada sección está enfocado en solventar algún tipo o codificación geométrica, operaciones o funciones geográficas, etc.

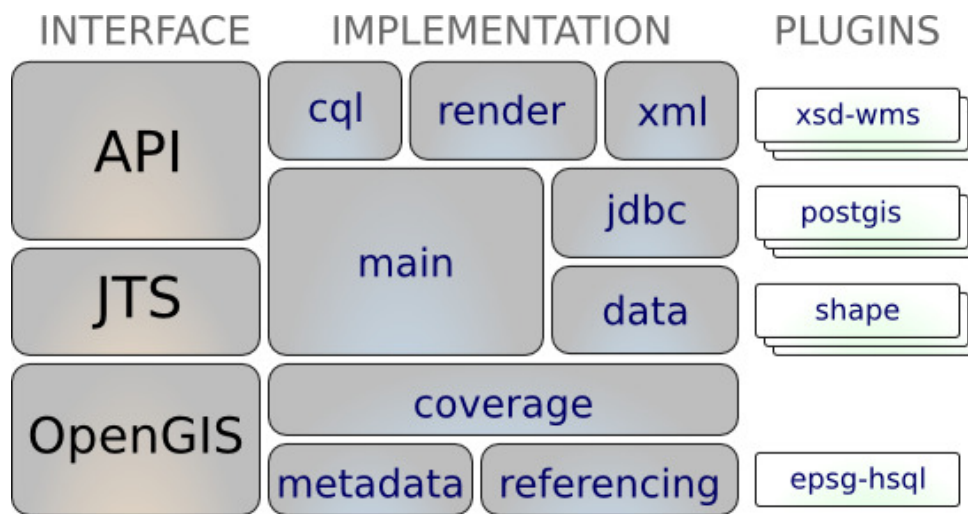


Figura 2.8: Estructura de GeoTools

Funcionalidades básicas

GeoTools es una potente herramienta que implementa varias funcionalidades útiles para el ámbito geográfico, a continuación se nombran algunas importantes aportaciones [10]:

1. Soporte integrado para geometrías
2. Atributos y filtros espaciales usando la especificación OGC Filter Encoding
3. Acceso a datos SIG¹⁷ en multitud de formatos de fichero y bases de datos espaciales
4. Soporte a una gran variedad de proyecciones cartográficas
5. Composición y visualización de mapas con simbología compleja

¹⁶Open Geospatial Consortium

¹⁷Sistemas de Información Geográfica

6. La tecnología de parseo/codificación proporciona conexiones a multitud de estándares OGC incluyendo GML, Filter, KML¹⁸, SLD¹⁹.
7. Las extensiones proporcionan soporte para grafos y redes (para encontrar rutas óptimas)

2.5.2. PostGIS

PostGIS es una extensión que convierte el sistema de base de datos PostgreSQL en una base de datos espacial (Ver Figura2.9). La combinación de ambos permite el almacenamiento, gestión y mantenimiento de datos espaciales [16].

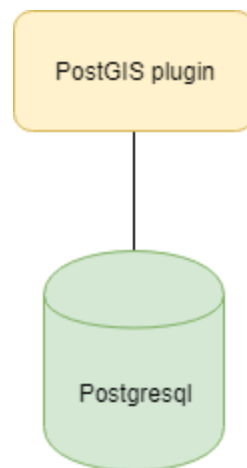


Figura 2.9: Estructura de PostGIS

Características

- PostGIS es software libre, tiene licencia GPL²⁰.
- Es compatible con los estándares de OGC²¹.
- Soporta tipos de datos espaciales, índices espaciales y tiene cientos de funciones espaciales.
- Permite importar y exportar datos a través de varias herramientas conversoras (shp2pgsql, pgsq2shp, ogr2ogr, dxf2postgis).

¹⁸KML es un lenguaje de marcado basado en XML para representar datos geográficos en tres dimensiones.

¹⁹(Styled Layer Descriptor) dentro de los Sistemas de Información Geográfica es un lenguaje estándar

²⁰GNU General Public License

²¹Open Geospatial Consortium



Selección de herramienta

En el análisis e investigación de GeoTools y Postgis para procesar información geoespacial se dio un enfoque al funcionamiento y prestaciones de cada una. Se tomó a GeoTools como herramienta que procesará los datos geoespaciales en la extensión GeoSPARQL. La razón por la que no se eligió postgis, es porque es una extensión para base de datos relacional (Postgresql) y trabaja con datos provenientes del mismo, mientras GeoTools es capaz de manejar datos espaciales provenientes de distintas fuentes.



Capítulo 3

Diseño e implementación de la extensión GeoSPARQL sobre CumulusRDF

En este capítulo se presenta el diseño e implementación de una extensión a CumulusRDF para el manejo de GeoSPARQL.

3.1. Diseño de la arquitectura

En la figura 3.1 se puede observar un diseño global, donde se encuentra integrada la arquitectura de la extensión GeoSPARQL a CumulusRDF. Este diseño parte de CumulusRDF que recibe una consulta GeoSPARQL, el cual envía al procesador de consultas y este devuelve dos resultados a CumulusRDF. La primera, es una consulta SPARQL, y la otra es una lista de condiciones geográficas extraídas de la consulta original. CumulusRDF procesa la consulta SPARQL recibiendo una lista de resultados y los envía al procesador de operaciones conjuntamente con las condiciones geográficas. El procesador de operaciones verifica que se cumplan las condiciones geográficas, tupla a tupla de la lista de resultados, entregando al final el resultado deseado.

A continuación se mostrará detalladamente el funcionamiento de cada componente de la extensión:

3.1.1. Procesador de consultas GeoSPARQL

Específicamente la función del procesador de consultas es tomar la consulta GeoSPARQL y extraer la parte geográfica, creando una consulta SPARQL y una lista de condiciones geográficas. La consulta SPARQL creada es fácilmente procesable por CumulusRDF. Este componente de la extensión utiliza Apache Jena, el cual es un framework

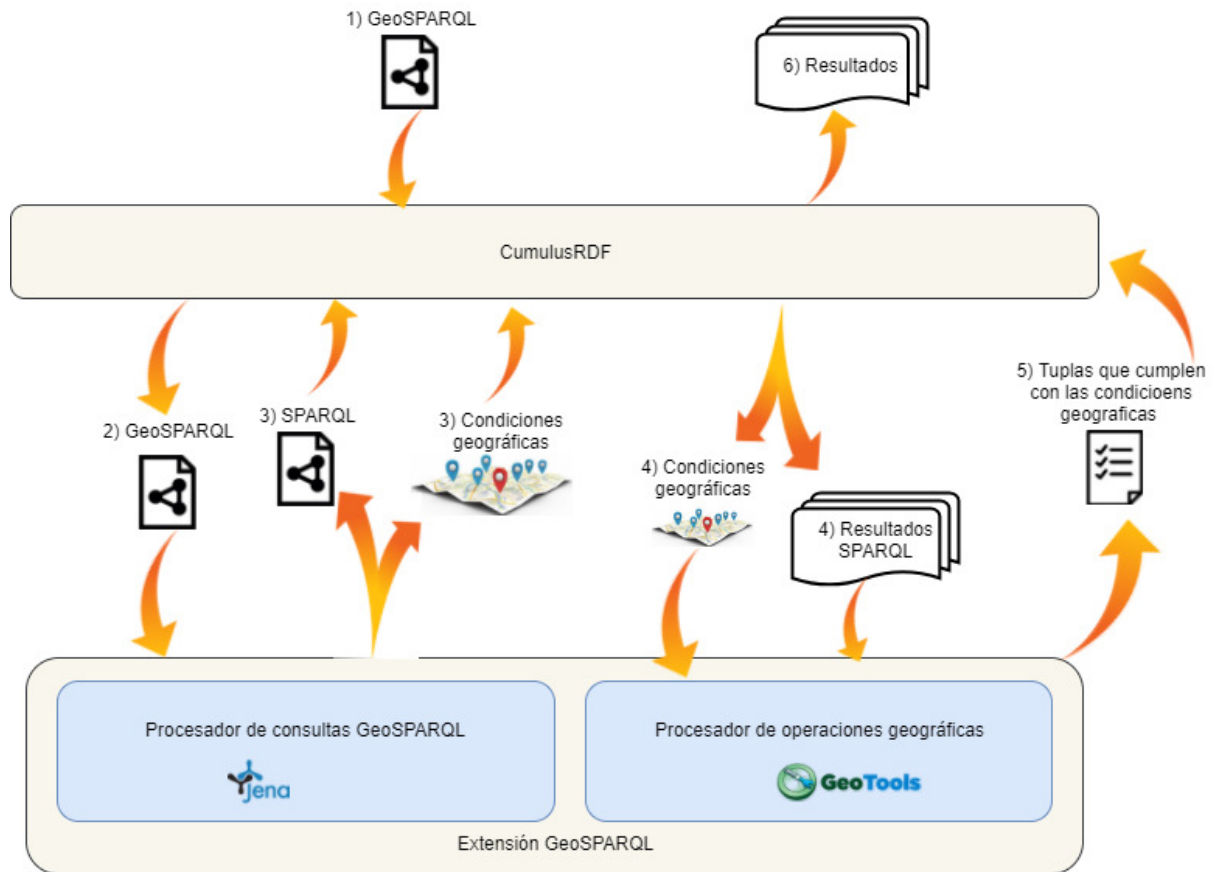


Figura 3.1: Arquitectura de la extensión GeoSPARQL con CumulusRDF

que se enfoca en leer, procesar y escribir RDF, también posee y motor de consultas compatible con SPARQL (Ver Figura 3.2).

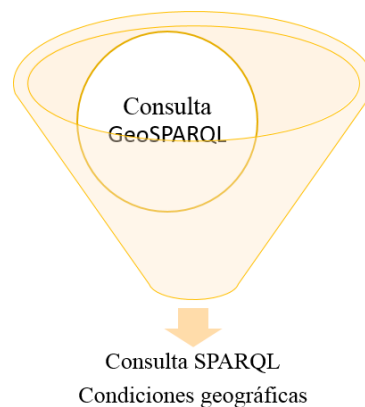


Figura 3.2: Ilustración del procesador de consultas



A continuación se desarrollará un ejemplo donde se mostrará paso a paso la transformación que sufre una consulta GeoSPARQL dentro de este procesador, hasta llegar a los resultados que entrega al final:

- **Paso 1:** El procesador recibe una consulta GeoSPARQL a través de CumulusRDF. La consulta que se presenta a continuación mostrará los 10 primeros municipios que pertenecen a la provincia de Madrid. Esta consulta hace uso de la función topológica binaria `sfContains`.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX geoes: <http://geo.marmotta.es/ontology#>
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX geof: <http://www.opengis.net/def/function/geosparql/>

SELECT DISTINCT ?labelMunicipios
WHERE {
  ?subject a <http://geo.marmotta.es/ontology#provincia>.
  ?subject rdfs:label "Madrid"@es.
  ?subject geoes:hasExactGeometry ?geo.
  ?geo geo:asWKT ?wkt.

  ?subject2 a <http://geo.marmotta.es/ontology#municipio>.
  ?subject2 rdfs:label ?labelMunicipios.
  ?subject2 geoes:hasExactGeometry ?geo2.
  ?geo2 geo:asWKT ?wkt2.

  FILTER (geof:sfContains(?wkt, ?wkt2))
}
ORDER BY ?labelMunicipios LIMIT 10
```

- **Paso 2:** El procesador mapea la consulta GeoSPARQL y separa todas las condiciones geográficas conjuntamente con sus variables, generando así una nueva consulta SPARQL. En esta nueva consulta generada omite los filtros de relaciones geográficas (`sfContains`, `sfWithin`, `sfCrosses` ...), si tiene un `LIMIT` también lo omite y finalmente agrega en el `SELECT` las variables que la función utiliza (`?wkt`, `?wkt2`, ...) y presenta el siguiente resultado:

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX geoes: <http://geo.marmotta.es/ontology#>
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX geof: <http://www.opengis.net/def/function/geosparql/>

SELECT DISTINCT ?labelMunicipios ?wkt ?wkt2
WHERE {
    ?subject a <http://geo.marmotta.es/ontology#provincia>.
    ?subject rdfs:label "Madrid"@es.
    ?subject geoes:hasExactGeometry ?geo.
    ?geo geo:asWKT ?wkt.

    ?subject2 a <http://geo.marmotta.es/ontology#municipio>.
    ?subject2 rdfs:label ?labelMunicipios.
    ?subject2 geoes:hasExactGeometry ?geo2.
    ?geo2 geo:asWKT ?wkt2.
}
ORDER BY ?labelMunicipios
```

- **Paso 3:** Una vez creada la nueva consulta SPARQL, se procede a crear una lista de condiciones geográficas (Ver Tabla 3.1). En la tabla están separadas la función y las variables que usará el procesador de operaciones.

Condición	Función	Variable 1	Variable 2
FILTER (geof:sfContains(?wkt, ?wkt2))	sfContains	wkt	wkt2
.	.	.	.
.	.	.	.
.	.	.	.

Tabla 3.1: Condiciones geográficas

La consulta SPARQL creada y las condiciones geográficas extraídas son enviadas a CumulusRDF para que puedan ser procesadas y enviadas al procesador de operaciones.

3.1.2. Procesador de operaciones geográficas

Este componente tiene la función de hacer cumplir las condiciones geográficas con los resultados de una consulta SPARQL. La lista de resultados obtenidos por parte de CumulusRDF es el resultado de haber procesado la consulta SPARQL creada por el procesador de consultas. Este procesador entrega un resultado final coherente, apegado a las

condiciones de la consulta GeoSPARQL original. Para lograr su objetivo este componente utiliza GeoTools.

Una vez obtenido respuestas de la consulta SPARQL, CumulusRDF enviará tupla a tupla al procesador de operaciones, conjuntamente con las condiciones geográficas (Ver Figura 3.3). Si la tupla cumple con toda lista de condiciones, CumulusRDF mostrará la tupla, caso contrario lo omitirá y continuará con la siguiente tupla, hasta que se cumpla el límite o termine con todas las tuplas de la lista de resultados.

Retomando el ejemplo que realizó con el procesador de consultas, en la tabla 3.2 se puede observar una posible lista de resultados que podría entregar CumulusRDF. El procesador de operaciones tomará conjuntamente cada tupla de la lista de resultados con la lista de condiciones geográficas, y para este caso verificará que la geometría wkt2 sea contenida por la geometría wkt. CumulusRDF mostrará únicamente la tupla que cumpla con todas las condiciones geográficas.

?labelMunicipios	?wkt	?wkt2
labelMunicipiosA	Polígono (-0.123123,1.123131 1.23233,-0.423423)	Polígono (-0.123123,1.123131 1.23233,-0.423423)
labelMunicipiosB	Polígono (-0.567567, -1.677465, -0.75673....)	Polígono (-0.123123,1.7564534 1.23233,-0.4434555)
.	.	.
.	.	.
.	.	.

Tabla 3.2: Lista de resultados SPARQL

3.2. Diseño e implementación del procesador de consultas

El procesador de consultas definido anteriormente se encuentra modelado por el diagrama de clases que se muestra en la figura 3.4. A continuación se detallarán las funciones principales y métodos principales de cada clase que constituyen el diagrama:

- **Component:** Almacena las relaciones topológicas binarias Simple Features. Recibe como parámetros, un elemento que contenga el filtro (element), la operación que se realizará (op), y las variables con las que operará (x, y).

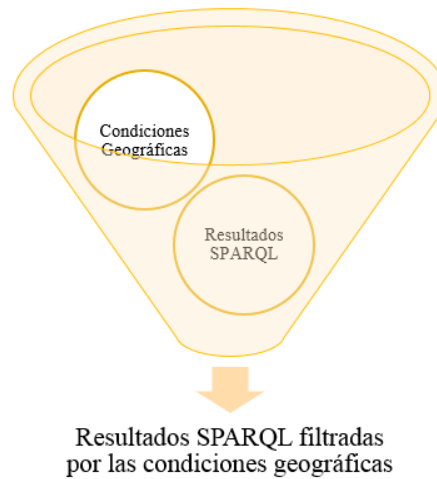


Figura 3.3: Ilustración del procesador de operaciones

- **ComponentSelect:** Almacena las funciones de filtro geoespaciales. Recibe como parámetros, la operación que se realizará (op), y las variables con las que operará (x, y) y por último la variable en la que almacenará el resultado (result).
- **QueryFragment:** Tiene la tarea de fragmentar la consulta GeoSPARQL, es decir, posee una serie de métodos para separar cada componente de la consulta y obtenerlas por separado. A continuación se describirán los métodos principales que posee:
 - **getQueryConditions():** Devuelve las condiciones no geográficas.
 - **getVars():** Devuelve las variables del select de la consulta original.
 - **getSelectGeoVars():** Devuelve las geográficas que se agregaran al select de la nueva consulta.
 - **getLimit():** Devuelve el limit de la consulta.
 - **isOrderBy():** Devuelve true si la consulta es ordenada.
 - **getOrderBy():** Devuelve una lista de variables por las que los resultados serán ordenados.
 - **getGeoQueryConditions():** Devuelve la lista de condiciones geográficas.
- **QueryBuilder:** Esta clase instancia la clase QueryFragment para construir una consulta SPARQL y devolver una lista de condiciones geográficas. A continuación se describirá los métodos principales que posee:
 - **buildSimpleQuery():** Construye la consulta SPARQL.
 - **getListComponents():** Devuelve la lista de condiciones geográficas de tipo topológicas binarias Simple Features.

- **getComponentes_select():** Devuelve la lista de condiciones geográficas de tipo filtro geoespaciales.
- **QueryManager:** Esta es una clase gestor que instancia la clase QueryBuilder y utiliza sus métodos.

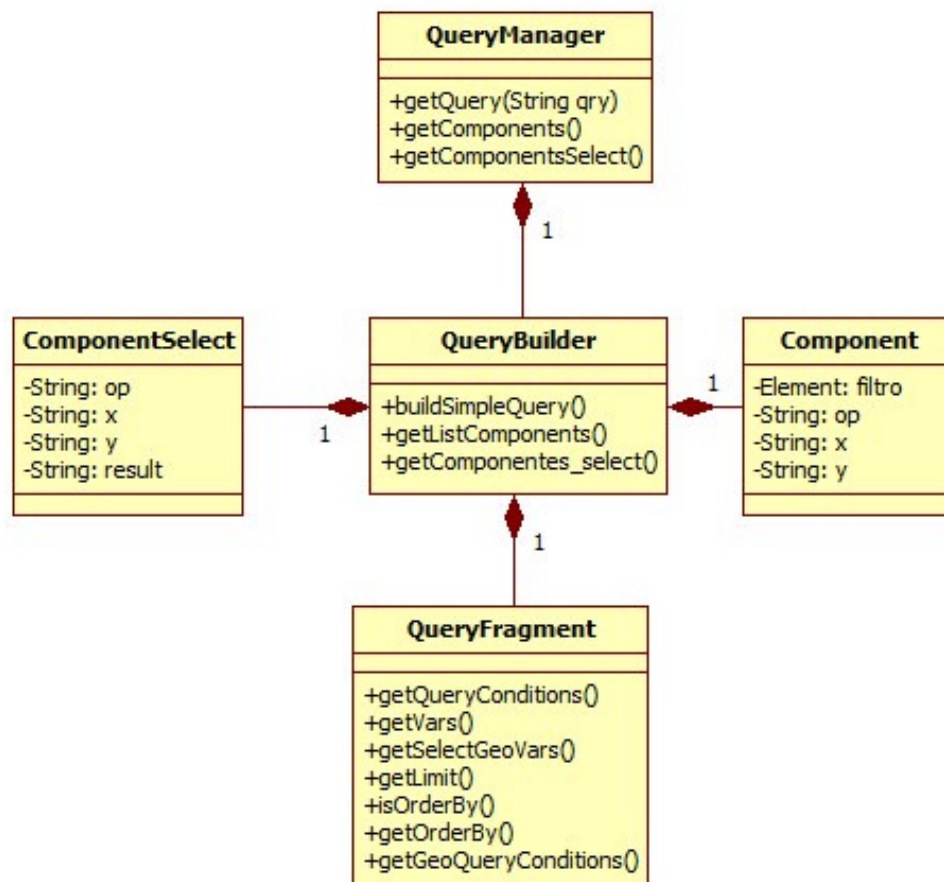


Figura 3.4: Diagrama de clases del procesador de consultas



A continuación se muestra paso a paso el funcionamiento de este componente:

1. QueryManager recibe la consulta GeoSPARQL por parte de CumulusRDF.
2. QueryManager envía la consulta GeoSPARQL a QueryFragment a través de la clase QueryBuilder.
3. QueryFragment recibe la consulta y la fragmenta en distintos componentes como; en condiciones geográficas y no geográficas, componentes del select, variables geográficas, etc.
4. QueryBuilder construye la consulta SPARQL con en el método buildSimpleQuery() con la ayuda de los métodos declarados en la clase QueryFragment.
5. QueryBuilder crea una lista de condiciones geográficas de tipo ComponentSelect o Component.
6. Por último, QueryManager extrae la consulta SPARQL y las condiciones geográficas de la clase QueryBuilder proporcionada los métodos buildSimpleQuery(), getListComponents() y getComponents_select().

3.3. Diseño e implementación del procesador de operaciones

Para esta sección se utilizó el patrón de diseño de software strategy el cual permite disponer de varios métodos para resolver un problema y elegir el adecuado en tiempo de ejecución. Los diagramas que se ven en las figuras 3.5 y 3.6.

A continuación se detallarán las funciones principales de cada clase:

- **IOperatios o IOperationsGeometry:** Estas interfaces contiene los métodos calculate los cuales devuelven valores booleanos y geométricos respectivamente.
- **SfContains, Union ... :** Estas clases implementan los métodos de las interfaces, es decir devuelven los resultados después operar con dos geometrías.
- **Operations u OperationsGeometry:** Poseen un método llamado caculate que recibe las geometrías en formato String y los transforma a objetos Geometry para que puedan ser procesados por una instancia de una clase que implemente la interfaz IOperations y este devuelva el resultado deseado.

Por último, el diagrama de clases de la figura 3.7 une los diagramas las operaciones de relaciones topológicas Simple Features y de las funciones filtro espaciales. Los métodos de la clases EjectCalculus; topologicalRelactions y geometry instancian la clase correspondiente que resolverá operación geográfica con las geometrías acorde al tipo de función que reciba.

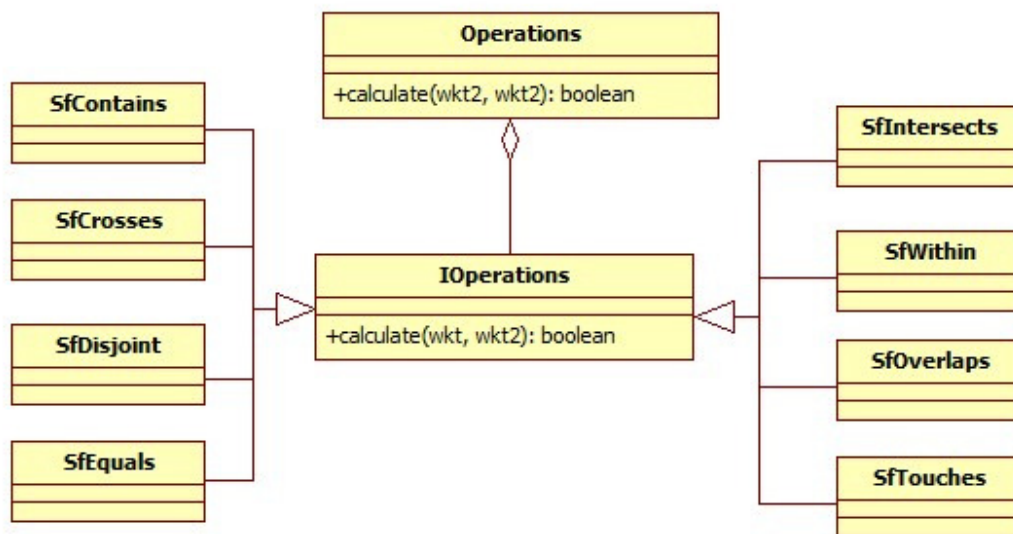


Figura 3.5: Diagrama de clases de las relaciones topológicas binarias Simple Features

A continuación se muestra paso a paso el funcionamiento de este componente:

1. EjectCalculus recibe una tupla y las condiciones geográficas.
2. EjectCalculus toma las condiciones geográficas y dependiendo el tipo de condiciones instanciará clases SfContains, SfCrosses, ... de tipo Operations u OperationsGeometry.
3. EjectCalculus envía la tupla a la clase instanciada.
4. La clase instanciada (SfContains, SfCrosses, ...) opera la tupla y devuelve un resultado.
5. EjectCalculus recibe el resultado de la clase instanciada, y este puede ser un Booleano o Geométrico.
6. EjectCalculus envía el resultado a CumulusRDF

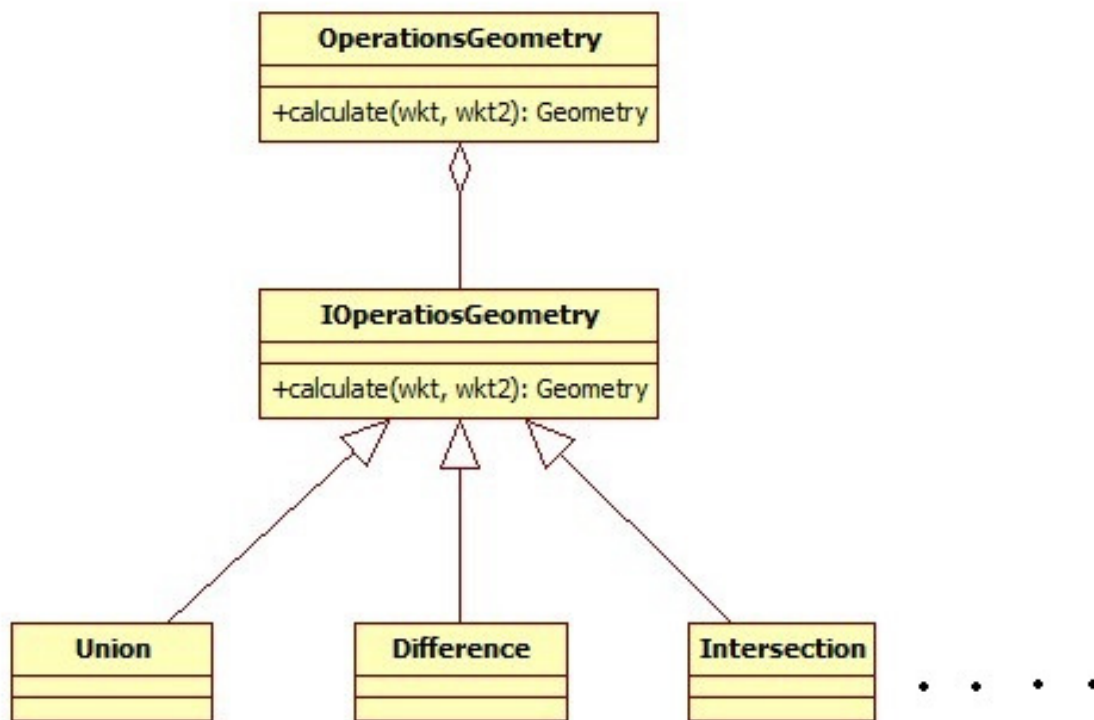


Figura 3.6: Diagrama de clases de las funciones filtro geoespaciales

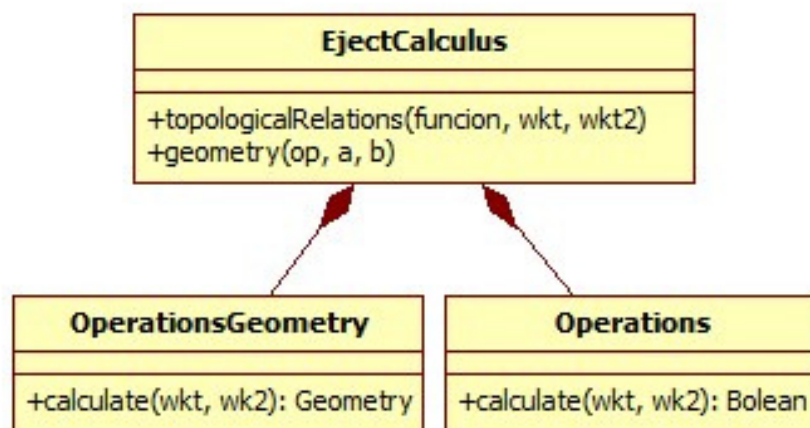


Figura 3.7: Diagrama que une las relaciones topológicas binarias Simple Features y las ficciones filtro geoespaciales.

3.4. Integración GeoSPARQL con CumulusRDF

El módulo GeoSPARQL consta de dos componentes, el procesador de consultas y el procesador de operaciones geográficas. Para integrarlo a CumulusRDF, se modificó el módulo Standalone originario de CumulusRDF y se agregó una clase llamada GQuery. Este se encarga de llamar a los métodos de cada procesador y de CumulusRDF con el fin de que interactúen mutuamente (Ver figura 3.8). A continuación se detalla paso a paso el funcionamiento de esta clase:

1. Recibe la consulta GeoSPARQL.
2. Envía la consulta GeoSPARQL a procesador de consultas (QueryManager).
3. Recibe una consulta SPARQL y una lista de condiciones geográficas por parte del procesador de consultas (QueryManager).
4. Envía la consulta SPARQL a CumulusRDF para que sea procesado.
5. Recibe una lista de resultados por parte de CumulusRDF después de procesar la consulta SPARQL.
6. Envía una tupla de la lista de resultados al procesador de operaciones (EjectCalculus), conjuntamente con la lista de condiciones geográficas.
7. Recibe true, false o una geometría por parte del procesador de operaciones (EjectCalculus) después de aplicar las condiciones geográficas a la tupla de la lista de resultados.
8. Muestra al usuario el resultado, si la tupla ha cumplido con todas las condiciones geográficas, caso contrario lo omite. Se repetirá el paso 6, 7 y 8 hasta que se cumpla el límite de la consulta o se haya terminado todas las tuplas de la lista de resultados.

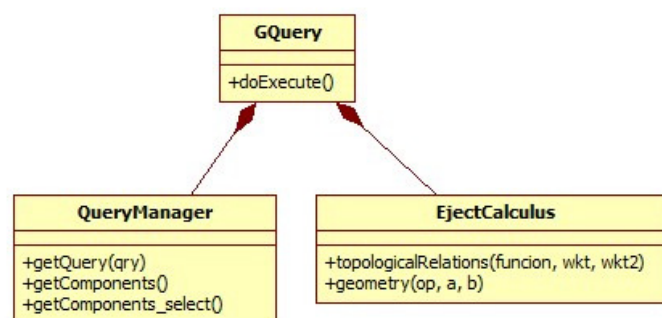


Figura 3.8: Diagrama que une la integración de los procesadores a CumulusRDF



Capítulo 4

Evaluación de resultados

Este capítulo presenta el proceso que se llevó a cabo para probar la efectividad de la extensión, ejecutándolo en un clúster y comparando sus resultados con los resultados de las consultas de la implementación GeoSPARQL en Apache Marmotta.

4.1. Equipo

El equipo de pruebas fue constituido por un clúster de cuatro nodos. Cada nodo tuvo las siguientes características:

- **Sistema Operativo:** Ubuntu 17.04 de 64-bits
- **Procesador:** Core i7
- **RAM:** 12 Gigas
- **Disco Duro:** 500Gigas

Para la configuración de CumulusRDF Multi-Node se utilizó la versión 2.2.9 de Apache Cassandra (Ver Apéndices D y E).

4.2. Datos de prueba

Para este proyecto, se tomaron fuentes de datos provenientes de linkedgeodata ¹ y sobre todo, los datos de prueba de la implementación GeoSPARQL realizada sobre Apache Marmotta ² lo cuales se puede encontrar en su página oficial ³. Los datos de prueba describen la información geoespacial de provincias, municipios y ríos de España en RDF.

El método de carga de datos para el clúster se encuentra definido en el apéndice E.

¹<http://downloads.linkedgeodata.org/releases/>

²(Linked Media Framework project) proporciona la lectura y escritura completa de SPARQL.

³<https://wiki.apache.org/marmotta/MARMOTTA-584/UserDocumentation>



4.3. Ejecución de las consultas de pruebas

Las consultas que se utilizaron para la prueba de la herramienta desarrollada en este proyecto fueron las consultas de prueba de la implementación GeoSPARQL de Apache Marmotta definidas en su página oficial. En dicha página se puede encontrar consultas con cada función geoespacial definida por la OGC ⁴. Para ejemplificar el funcionamiento a continuación se mostrará a ejecución de las siguientes consultas:

1. Obtener las primeras diez ciudades que contiene la provincia de Madrid.

- **Función:** Contains

- **Consulta GeoSPARQL:**

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX geoes: <http://geo.marmotta.es/ontology#>
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX geof: <http://www.opengis.net/def/function/geosparql/>

SELECT DISTINCT ?labelMunicipios

WHERE {
  ?subject a <http://geo.marmotta.es/ontology#provincia>.
  ?subject rdfs:label "Madrid"@es.
  ?subject geoes:hasExactGeometry ?geo.
  ?geo geo:asWKT ?wkt.

  ?subject2 a <http://geo.marmotta.es/ontology#municipio>.
  ?subject2 rdfs:label ?labelMunicipios.
  ?subject2 geoes:hasExactGeometry ?geo2.
  ?geo2 geo:asWKT ?wkt2.

  FILTER (geof:sfContains(?wkt, ?wkt2))
}
ORDER BY ?labelMunicipios
LIMIT 10
```

- **Resultados:** En este punto se muestra los resultados obtenidos y se los compara. En la figura 4.1 se puede ver los resultados de ambas herramientas, y evidentemente al comparar los resultados son los mismos.

⁴Open Geospatial Consortium

a)

13:10:11,731 INFO 110: [labelMunicipios="Ajalvir"@es]

13:10:12,288 INFO 216: [labelMunicipios="Alcalá de Henares"@es]

13:10:12,409 INFO 243: [labelMunicipios="Alcobendas"@es]

13:10:12,516 INFO 266: [labelMunicipios="Alcorcón"@es]

13:10:13,147 INFO 395: [labelMunicipios="Algete"@es]

13:10:13,767 INFO 519: [labelMunicipios="Alpedrete"@es]

13:10:13,998 INFO 574: [labelMunicipios="Anchuelo"@es]

13:10:14,511 INFO 696: [labelMunicipios="Arganda del Rey"@es]

13:10:14,782 INFO 762: [labelMunicipios="Arroyomolinos"@es]

13:10:15,976 INFO 1027: [labelMunicipios="Becerril de la Sierra"@es]

BUILD SUCCESS

b)

<i>labelMunicipios</i>
Ajalvir @es
Alcalá de Henares @es
Alcobendas @es
Alcorcón @es
Algete @es
Alpedrete @es
Anchuelo @es
Arganda del Rey @es
Arroyomolinos @es
Becerril de la Sierra @es

Figura 4.1: a) Resultado de la implementación GeoSPARQL sobre CumulusRDF. b) Resultado de la implementación GeoSPARQL sobre Apache Marmotta.

2. Obtener la unión de las provincias de Madrid y Barcelona

- **Función:** Union
- **Consulta GeoSPARQL:**

```

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX geoes: <http://geo.marmotta.es/ontology#>
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX geof: <http://www.opengis.net/def/function/geosparql/>

SELECT DISTINCT ?wktA ?wktB (geof:union(?wktA,?wktB) as ?union)

WHERE {
  ?subjecta <http://geo.marmotta.es/ontology#provincia>.
  ?subject rdfs:label "Madrid"@es.
  ?subject geoes:hasExactGeometry ?geo.
  ?geo geo:asWKT ?wktA.

  ?subject2 a <http://geo.marmotta.es/ontology#provincia>.
  ?subject2 rdfs:label "Barcelona" @es.
  ?subject2 geoes:hasExactGeometry ?geo2.
  ?geo2 geo:asWKT ?wktB.

}

```

- **Resultado:** El resultado que entregó CumulusRDF fue una secuencia de puntos en formato WKT, pero, para ejemplificar de mejor manera, se tomó los puntos y se graficó con una herramienta llamada QGIS. De igual manera que el ejemplo anterior, en la figura 4.2 se puede observar los resultados de ambas herramientas y de igual modo, evidentemente los resultados son los mismos.

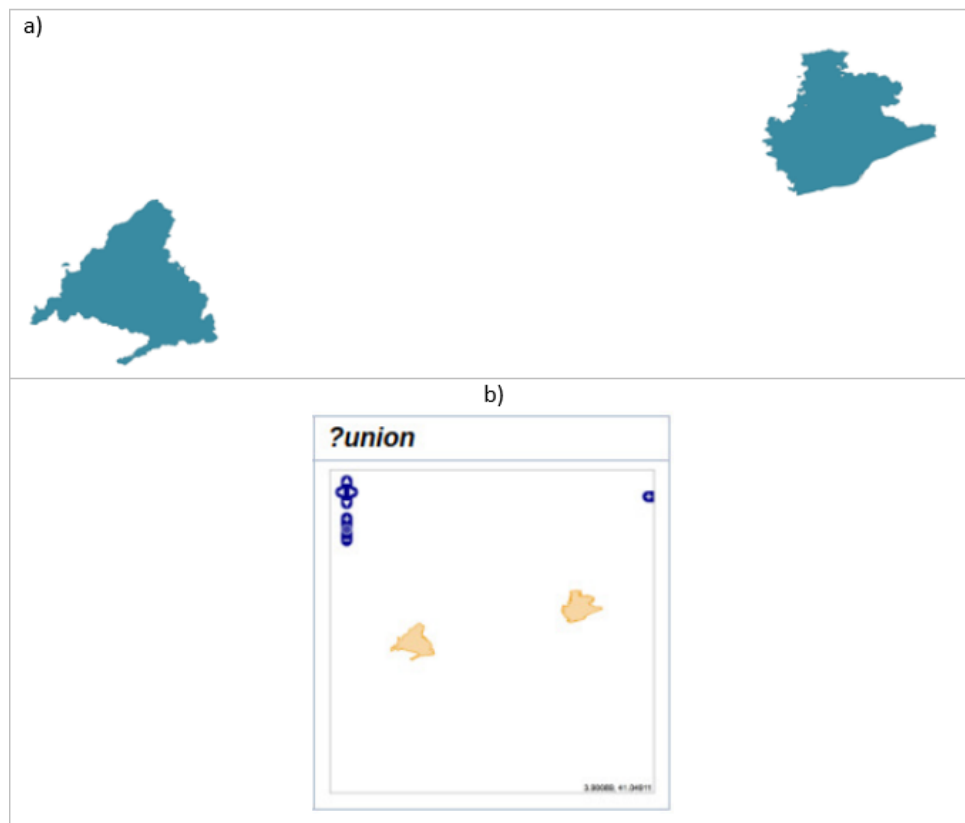


Figura 4.2: a) Resultado de la implementación GeoSPARQL sobre CumulusRDF. b) Resultado de la implementación GeoSPARQL sobre Apache Marmotta.

Como se pueden observar, los resultados de ambas herramientas para las mismas consultas, son iguales. En conclusión, se prueba que el funcionamiento de la extensión GeoSPARQL desarrollada en este proyecto fue exitoso.



Conclusiones y trabajos futuros

El presente proyecto de graduación tuvo como objetivo implementar una extensión que de soporte GeoSPARQL sobre una base de datos NoSQL-RDF. El desarrollo de este proyecto partió de la necesidad de procesar grandes volúmenes de información geográfica RDF en un sistema distribuido a través de tecnologías NoSQL. Por la razón mencionada, se realizó una investigación de dichas tecnologías que fueran capaces de realizarlo. Entre las herramientas analizadas, se encontró a CumulusRDF que puede almacenar información RDF dentro de Apache Cassandra y GeoTools que es capaz de procesar datos geográficos. Como se puede notar, existen herramientas capaces, pero no trabajan conjuntamente. El diseño de la extensión partió de fusionar dichas tecnologías, para obtener así, el procesamiento de datos geográficos dentro de la herramienta NoSQL-RDF. La implementación del diseño creó una nueva herramienta, con la capacidad de tomar consultas GeoSPARQL y procesarlas dentro de un CumulusRDF. La extensión es capaz de resolver las relaciones topológicas binarias Simple Features y tres funciones filtro geoespaciales (union, difference e intersection). Para verificar el funcionamiento y efectividad, CumulusRDF + Extensión GeoSPARQL fue ejecutada en un clúster y los resultados obtenidos fueron comparados con los resultados de la implementación GeoSPARQL realizada sobre Apache Marmotta, siendo estos exactamente iguales.

Una de las limitaciones de este proyecto es que se implementó únicamente las relaciones topológicas binarias Simple Features y tres funciones filtro geoespaciales, sin embargo el estándar GeoSPARQL definido por la OGC engloba muchos más vocabularios y funciones. A futuro se piensa continuar con la implementación de los demás vocabularios y funciones filtro, para al final llegar a obtener una herramienta más robusta y completa. Una segunda limitación es la imposibilidad de procesar GeoSPARQL de forma nativa por parte de CumulusRDF, de igual manera a futuro se piensa plantear e implementar una solución para que CumulusRDF pueda procesar consultas GeoSPARQL sin tener la necesidad de utilizar alguna librería o un componente externo, llegando a mejorarlo y potenciarlo ampliamente.



Apéndices



Apéndice A

Instalación Single-Node de Apache Cassandra

A continuación se muestra la descarga e instalación de la versión 2.2.9 de Apache Cassandra:

Descarga

- `wget http://www.apache.org/dyn/closer.lua/cassandra/2.2.9/apache-cassandra-2.2.9-bin.tar.gz`

Descompresión

- `tar -zxf apache-cassandra-2.2.9-bin.tar.gz`

Creación de carpetas para librerías y logs

- `sudo mkdir /var/lib/cassandra`
- `sudo mkdir /var/log/cassandra`
- `sudo chown -R $USER:$GROUP /var/lib/cassandra`
- `sudo chown -R $USER:$GROUP /var/log/cassandra`

Configuración de las variables de Cassandra

- `export CASSANDRA_HOME=~/.cassandra`
- `export PATH=$PATH:$CASSANDRA_HOME/bin`



Ejecución

Correr los siguientes comandos en la terminal:

- Ejecutar Cassandra: `sudo sh ~/cassandra/bin/cassandra`
- Ejecuta la consola de Cassandra: `sudo sh ~/cassandra/bin/cqlsh`

Una vez ejecutado todos los comandos cassandra se ejecutará sin ningún problema, y se podrá observar en la terminal, tal como se muestra en la figura A.1

```
Connected to Test Cluster at 127.0.0.1:9042.  
[cqlsh 5.0.1 | Cassandra 2.2.9 | CQL spec 3.3.1 | Native protocol v4]  
Use HELP for help.  
cqlsh> █
```

Figura A.1: Consola de Cassandra



Apéndice B

Instalación de CumulusRDF

A continuación se muestra el procedimiento que se tomó para poner en funcionamiento a CumulusRDF. La versión más reciente se encuentra disponible en GitHub el cual es una plataforma de desarrollo colaborativo de software para alojar proyectos.

Clonación

Con la ayuda de Github se clonó el repositorio orinal a un repositorio personal para poder modificarlo sin inconvenientes, el mismo se encuentra alojado en:

- <https://github.com/fheragui/geocumulusrdf>

Descarga y compilación de fuentes

Con la ayuda de NetBeans 8.2 se descargaron las fuentes del repositorio personal en GitHub. Una vez descargado CumulusRDF, previo a la compilación se seleccionó el archivo pom.xml de más alto nivel (Ver Figura B.1) y se realizaron las siguientes modificaciones:

- Se eligió el perfil de la versión 2.x de cassandra (Ver Figura B.2).
- Se comentó el paquete cumulusrdf-integration-tests (Ver figura B.3)

Una vez configurado, con la ayuda de NetBeans 8.2 se procedió a compilar el proyecto con sus dependencias maven, obteniendo un resultado deseado (Ver Figura B.4).

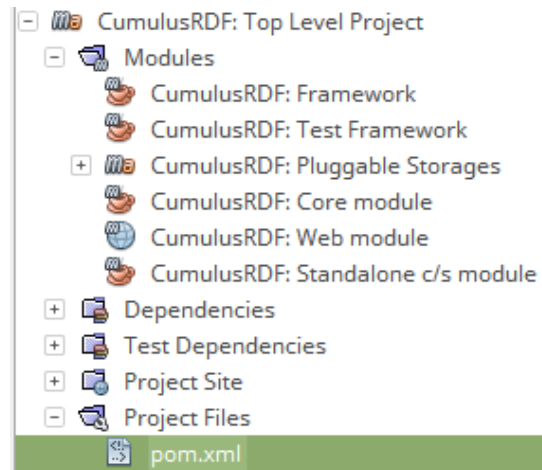


Figura B.1: Estructura general del proyecto

```
<profile>
  <id>cassandra2x-cql-full-tp-index</id>
  <properties>
    <cassandra.version>2.2.9</cassandra.version>
  </properties>
</profile>
```

Figura B.2: Perfil seleccionado para la compilación

```
<modules>
  <module>cumulusrdf-framework</module>
  <module>cumulusrdf-test-framework</module>
  <module>cumulusrdf-pluggable-storage</module>
  <module>cumulusrdf-core</module>
  <module>cumulusrdf-web-module</module>
  <module>cumulusrdf-standalone</module>
  <!--<module>cumulusrdf-integration-tests</module>-->
</modules>
```

Figura B.3: Orden de compilación de módulos

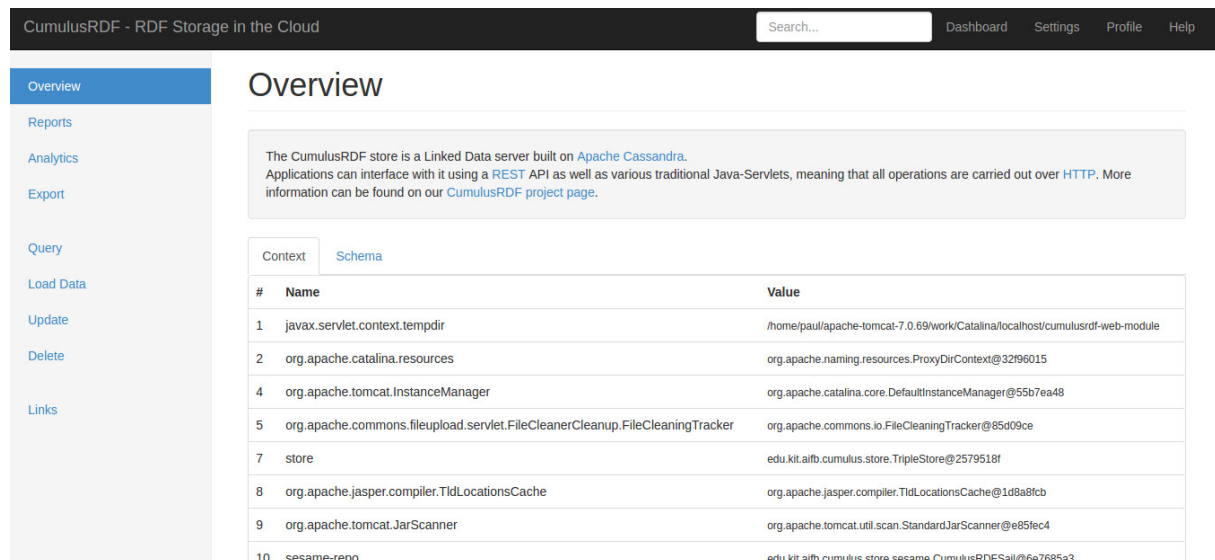
Ejecución

La ejecución CumulusRDF tiene dos posibilidades; la una corresponde al Standalone que lo podemos correr en la consola, y la otra es utilizar el ambiente web, llamado Web Module.

```
CumulusRDF: Top Level Project ..... SUCCESS [0.823s]
CumulusRDF: Framework ..... SUCCESS [2.937s]
CumulusRDF: Test Framework ..... SUCCESS [0.143s]
CumulusRDF: Pluggable Storages ..... SUCCESS [0.005s]
CumulusRDF: Cassandra 1.2.x (using Hector) ..... SUCCESS [1.116s]
CumulusRDF: Cassandra 2.x (using CQL) ..... SUCCESS [0.448s]
CumulusRDF: Core module ..... SUCCESS [1.311s]
CumulusRDF: Web module ..... SUCCESS [8.070s]
CumulusRDF: Standalone c/s module ..... SUCCESS [10.744s]
-----
BUILD SUCCESS
-----
```

Figura B.4: Módulos compilados

Para ejecutar el Web Module se debe instalar un servidor de aplicaciones, que en este caso se utilizó Apache Tomcat versión 7.x. una vez cargada la aplicación se accedió a la siguiente dirección <http://localhost:8080/cumulusrdf-web-module> y se observará la página de inicio de CumulusRDF (Ver Figura B.5), además se puede observar las opciones de carga y consulta de grafos RFD. Cabe recalcar que Apache Cassandra debe estar ejecutándose.



#	Name	Value
1	javax.servlet.context.tempdir	/home/paul/apache-tomcat-7.0.69/work/Catalina/localhost/cumulusrdf-web-module
2	org.apache.catalina.resources	org.apache.naming.resources.ProxyDirContext@32f96015
4	org.apache.tomcat.InstanceManager	org.apache.catalina.core.DefaultInstanceManager@55b7ea48
5	org.apache.commons.fileupload.servlet.FileCleanerCleanup.FileCleaningTracker	org.apache.commons.io.FileCleaningTracker@85d09ce
7	store	edu.kit.ai.fh.cumulus.store.TripleStore@2579518f
8	org.apache.jasper.compiler.TldLocationsCache	org.apache.jasper.compiler.TldLocationsCache@1d8a8fcb
9	org.apache.tomcat.util.scan.StandardJarScanner	org.apache.tomcat.util.scan.StandardJarScanner@e85fec4
10	sesame-repo	edu.kit.ai.fh.cumulus.store.sesame.CumulusRDFSail@6e7685a3

Figura B.5: Página de inicio de CumulusRDF

La opción Standalone trabaja en la terminal, y para cargar datos o ejecutar una consulta se utiliza parámetros. Todos los parámetros y opciones se encuentran claramente en la documentación de CumulusRDF (<https://github.com/cumulusrdf/cumulusrdf/wiki/CLI>).



Apéndice C

Configuración de CumulusRDF para implementación de la extensión GeoSPARQL

A continuación se detallará paso a paso la configuración de CumulusRDF que se realizó para implementar la extensión GeoSPARQL.

Creación y configuración

Con la ayuda de NetBeans se creó un proyecto maven al cual se nombró Geography y para integrarlo al proyecto general de CumulusRDF se modificó el archivo pom.xml de alto nivel, agregando el nuevo módulo (Ver Figura C.1) quedando integrado completamente a CumulusRDF (Ver Figura C.2).

```
<modules>
  <module>cumulusrdf-framework</module>
  <module>cumulusrdf-test-framework</module>
  <module>cumulusrdf-pluggable-storage</module>
  <module>cumulusrdf-core</module>
  <module>cumulusrdf-web-module</module>
  <module>cumulusrdf-standalone</module>
  <module>Geography</module>
  <!-- <module>cumulusrdf-integration-tests</module> -->
</modules>
```

Figura C.1: Modificación del archivo pom.xml

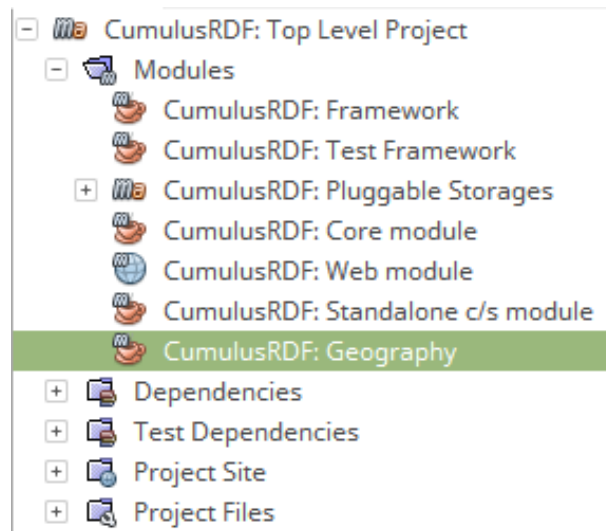


Figura C.2: Estructura del proyecto CumulusRDF

Agregación de dependencias

Una vez creado e integrado el módulo Geography, se procedió a integrar todas las dependencias en el archivo pom.xml de Geography, las cuales fueron utilizadas para la implementación GeoSPARQL.

Para manipular y construir de consultas SPARQL se utilizó Apache Jena (Ver Figura C.3), y para las operaciones geográficas, GeoTools 17.1 (Ver Figura C.4), para descargar las dependencias de GeoTools es necesario agregar un repositorio adicional (Ver figura C.5).

```
<dependency>
  <groupId>org.apache.jena</groupId>
  <artifactId>jena-arq</artifactId>
  <version>3.3.0</version>
  <type>jar</type>
</dependency>
```

Figura C.3: Dependencia Jena

Una vez creado el módulo Geography y agregado todas las dependencias, CumulusRDF queda listo para implementar GeoSPARQL.



```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.11</version>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>org.geotools</groupId>
  <artifactId>gt-shapefile</artifactId>
  <version>17.1</version>
</dependency>
```

Figura C.4: Dependencia GeoTools

```
<repositories>
  <repository>
    <id>osgeo</id>
    <name>Open Source Geospatial Foundation Repository</name>
    <url>http://download.osgeo.org/webdav/geotools/</url>
  </repository>
</repositories>
```

Figura C.5: Repositorio GeoTools



Apéndice D

Configuración del clúster Multi-Node de Apache Cassandra

A continuación se muestra la configuración del clúster de Apache Cassandra (realizar el mismo procedimiento en todos los nodos del clúster).

Descarga

- `wget http://www.apache.org/dyn/closer.lua/cassandra/2.2.9/apache-cassandra-2.2.9-bin.tar.gz`

Descomprimir el contenido en la carpeta `~/cassandra`

- `tar -zxf apache-cassandra-2.2.9-bin.tar.gz`

Creación de carpetas para librerías y logs

- `sudo mkdir /var/lib/cassandra`
- `sudo mkdir /var/log/cassandra`
- `sudo chown -R $USER:$GROUP /var/lib/cassandra`
- `sudo chown -R $USER:$GROUP /var/log/cassandra`

Configuración de las variables de Cassandra

- `export CASSANDRA_HOME=~/cassandra`
- `export PATH=$PATH:$CASSANDRA_HOME/bin`



Borrar el contenido de la carpeta data

- `sudo rm -r cassandra/data/*`

Modificar el siguiente archivo `~/cassandra/conf/cassandra.yaml`

Reemplazar `your_server_ip` con la dirección IP del servidor que se está trabajando.
En `- seeds`: agregar la listas de todas las direcciones IP que se involucran en el clúster.

```
cluster_name: 'CumulusrdfCluster'

. . .

seed_provider:
  - class_name: org.apache.cassandra.locator.SimpleSeedProvider
    parameters:
      - seeds: "your_server_ip,your_server_ip_2,...your_server_ip_n"

. . .

listen_address: your_server_ip

. . .

rpc_address: your_server_ip

. . .

endpoint_snitch: GossipingPropertyFileSnitch

. . .

Agregar al final del contenido del archivo:

auto_bootstrap: false
```

Ejecución

Ejecutar los siguientes comandos en cada nodo:

- Ejecutar Cassandra: `sudo sh ~/cassandra/bin/cassandra`



- Ejecuta la consola de Cassandra: `sudo sh ~/cassandra/bin/cqlsh [your_server_ip] [port]` (Ver Figura D.1)

```
estudiante@CCING-6-00:~/cassandra/bin$ sudo sh ~/cassandra/bin/cqlsh 172.16.147.59 9042
Connected to CumulusrdfCluster at 172.16.147.59:9042.
[cqlsh 5.0.1 | Cassandra 2.2.9 | CQL spec 3.3.1 | Native protocol v4]
Use HELP for help.
cqlsh>
```

Figura D.1: Consola de Apache Cassandra

Verificación del estado del clúster:

```
estudiante@CCING-6-00:~/cassandra/bin$ ./nodetool status
Datacenter: datacenter1
=====
Status=Up/Down
-- State=Normal/Leaving/Joining/Moving
-- Address          Load        Tokens      Owns (effective)  Host ID                               Rack
UN 172.16.147.96     102,24 KB   256         51,7%             26834e61-765a-4aae-8fa5-55de795ac247 rack1
UN 172.16.147.150    67,77 KB   256         49,5%             030e9f5f-ed0f-4e6e-b631-3cf25063b9e2 rack1
UN 172.16.147.88     102,22 KB   256         51,0%             c1073c4b-7ce8-497d-b688-c859b808d025 rack1
UN 172.16.147.59     84,61 KB   256         47,8%             8dc0daed-d59d-4f24-8269-5ef28008404a rack1
```

Figura D.2: Estado del cluster



Apéndice E

Definición de carga y consulta de datos GeSPARQL para CumulusRDF

Configuración de CumulusRDF

Una de las características de CumulusRDF, es su funcionamiento en modo Single-Node ¹ o Multi-Node ². Para que funcione como Single-Node, Apache Cassandra debe estar configurada como Single-Node definido en el apéndice A y de la misma manera si lo queremos en Multi-Node definido en el apéndice D. Cabe recalcar que CumulusRDF también necesita cierta configuración.

Single-Node y Multi-Node

Es necesario modificar el archivo `cumulusRDF-default.yaml`. En la figura E.1 se muestra la configuración realizada para single-node, donde, *cassandra-hosts* recibe la especificación de la dirección IP donde se está ejecutando Apache Cassandra.

En la figura E.2 se puede observar la configuración multi-node de CumulusRDF. Se destaca *cassandra-hosts* que recibe todos los IP's del clúster de Cassandra y *cassandra-replication-factor* que recibe el factor de replicación (números e nodos que posee el clúster).

Carga de datos

Para cargar los datos, la documentación de CumulusRDF proporciona el patrón a utilizar, el cual se muestra a continuación:

- `java -cp cumulusRDF.jar edu.kit.aifb.cumulus.cli.Cirrus load [opciones...]`

¹Trabajan sobre un solo servidor

²Trabajan sobre un cluster de servidores



```

cassandra-hosts: "172.16.147.88"
# Keyspace in Cassandra cluster.
# This is valid for Cassandra 1.2.x and Cassandra 2.x
# By using several keyspaces, multiple CumulusRDF instances
# can store data in the same Cassandra cluster.
# Default value: "KeyspaceCumulus".
cassandra-keyspace: "KeyspaceCumulus"

# Number of replicas across the cassandra cluster.
# This is valid for Cassandra 1.2.x and Cassandra 2.x
# Default value: 1.
cassandra-replication-factor: 1

```

Figura E.1: Configuración Single-Node

```

cassandra-hosts: "172.16.147.88,172.16.147.96,172.16.147.150,172.16.147.59"
# Keyspace in Cassandra cluster.
# This is valid for Cassandra 1.2.x and Cassandra 2.x
# By using several keyspaces, multiple CumulusRDF instances
# can store data in the same Cassandra cluster.
# Default value: "KeyspaceCumulus".
cassandra-keyspace: "KeyspaceCumulus"

# Number of replicas across the cassandra cluster.
# This is valid for Cassandra 1.2.x and Cassandra 2.x
# Default value: 1.
cassandra-replication-factor: 4

```

Figura E.2: Configuración Multi-Node

Las lista de opciones que se puede utilizar se muestra en la tabla E.1. A continuación se muestra el proceso de carga de datos para este proyecto:

Patrón de carga de datos

- `load -i archivo.rdf -b 50`

Definición del patrón de consultas GeoSPARQL

De igual manera que la carga de datos, la documentación de CumulusRDF proporciona la información para ejecutar SPARQL. Para este proyecto se agregó una opción adicional para GeoSPARQL y se estableció el siguiente patrón:

- `java -cp cumulusRDF.jar edu.kit.aifb.cumulus.cli.Cirrus gquery [opciones...]`

La lista de opciones que se pueden usar, se muestra en la tabla E.2. A continuación se muestra el proceso de consulta para este proyecto:

Opciones	Descripción
-i	Nombre del archivo a leer
-n	Cassandra hosts, lista separada por comas ('host1: port1, host2: port2, ...') (localhost predeterminado: 9160).
-k	Cassandra keyspace (predeterminado KeyspaceCumulus)
-r	Factor de replicación (predeterminado: 1).
-s	Storage layout a usar (triple quad).
-b	Tamaño del lote - número de triples (valor predeterminado: 100s)
-f	Formato de serialización RDF: 'N-Triples', 'RDF / XML',

Tabla E.1: Opciones para la carga de datos

Opciones	Descripción
-k	Cassandra keyspace (predeterminado KeyspaceCumulus)
-n	Cassandra hosts, lista separada por comas ('host1: port1, host2: port2, ...') (localhost predeterminado: 9160).
-q	Consulta GeoSPARQL
-s	Storage layout a usar (triple quad).

Tabla E.2: Opciones para la consulta GeoSPARQL

Patrón de consulta de datos

- gquery -q "query"



Bibliografía

- [1] Sandra Aguirre, Joaquín Salvachúa, Juan Quemada, and Alberto Mozo. Uso del web semántico para la interoperabilidad semántica de recursos educativos en internet y redes p2p. *Telecom I+ D 2005*, 2005.
- [2] m-click.aero Aleksandar Balaban. OGC Engineering Report - Testbed-12 Aviation Semantics Engineering Report. <http://docs.opengeospatial.org/per/16-039.html>, 2016. [Modificado; 03-10-2017].
- [3] Tim Berners-Lee, James Hendler, Ora Lassila, et al. The semantic web. *Scientific american*, 284(5):28–37, 2001.
- [4] Pablo Castells. La web semántica. *Sistemas interactivos y colaborativos en la web*, pages 195–212, 2003.
- [5] Pablo Castells and Francisco Saiz. Ontologías para la web semántica. *La Web semántica: tecnologías y aplicaciones. Madrid: Escuela Politécnica Superior, Universidad Autónoma de Madrid*, 2003.
- [6] Philippe Cudré-Mauroux, Iliya Enchev, Sever Fundatureanu, Paul Groth, Albert Haque, Andreas Harth, Felix Leif Keppmann, Daniel Miranker, Juan F Sequeda, and Marcin Wylot. Nosql databases for rdf: an empirical evaluation. In *International Semantic Web Conference*, pages 310–325. Springer, 2013.
- [7] CumulusRDF. RDF in the cloud. <https://github.com/cumulusrdf/cumulusrdf/>, 2016. [Modificado; 04-14-2016].
- [8] Ana María García Martínez. Definición y estilo de los objetos de información digitales y metadatos para la descripción. *Boletín de la asociación Andaluza de bibliotecarios*, (63):23–47, 2001.
- [9] GeoTools. Geometry Relationships. <http://docs.geotools.org/stable/userguide/library/jts/relate.html>, 2017. [Modificado; 10-11-2017].
- [10] GeoTools. Supported Formats. <http://docs.geotools.org/stable/userguide/geotools.html#implemented-standards>, 2017. [Modificado; 10-11-2017].



- [11] Arantza Illarramendi, Jesús Bermúdez, Marta González, and Ana Isabel Torre-Bastida. Diseño de un repositorio rdf basado en tecnologías nosql. In *JISBD 2011. XVI Jornadas de Ingeniería del Software y Bases de Datos*, 2011.
- [12] Zoi Kaoudi and Ioana Manolescu. RDF in the Clouds: A Survey. *The International Journal on Very Large Databases*, June 2014.
- [13] Adriana Martín, Susana Beatriz Chávez, Nelson R Rodríguez, Adriana Valenzuela, and Maria A Murazzo. Bases de datos nosql en cloud computing. In *XV Workshop de Investigadores en Ciencias de la Computación*, 2013.
- [14] E Peis, Enrique Herrera-Viedma, Yusef Hassan-Montero, and Juan Carlos Herrera. Ontologías, metadatos y agentes: recuperación "semántica" de la información. 2003.
- [15] Matthew Perry and John Herring. Ogc geosparql-a geographic query language for rdf data. *OGC Implementation Standard*. Sept, 2012.
- [16] Postgis. Stable Branch User Documentation. <http://postgis.net/documentation/>, 2017. [Modificado; 10-10-2017].
- [17] Javier Gutiérrez Puebla. Los sistemas de información geográficos: origen y perspectivas. *Revista General de Información y Documentación*, 7(1):93, 1997.
- [18] Roshan Punnoose, Adina Crainiceanu, and David Rapp. Sparql in the cloud using rya. *Information Systems*, 48:181–195, 2015.
- [19] Arcgis Resources. Tres representaciones fundamentales de capas de información geográfica. <http://resources.arcgis.com/es/help/getting-started/articles/026n000000n000000.htm>, 2012. [Modificado; 09-27-2012].
- [20] Keilyn Rodríguez Perojo and Rodrigo Ronda León. Web semántica: un nuevo enfoque para la organización y recuperación de información en el web. *Acimed*, 13(6):0–0, 2005.
- [21] Alexander Castro Romero, Juan Sebastián González Sanabria, and Mauro Callejas Cuervo. Utilidad y funcionamiento de las bases de datos nosql. *Facultad de Ingeniería*, 21(33):21–32, 2012.
- [22] Apache Rya. Apache Rya (incubating). <https://rya.apache.org/>, 2017. [Modificado; 03-10-2017].
- [23] Alexander Schätzle, Martin Przyjaciół-Zablocki, Thomas Hornung, and Georg Lausen. Pigsparql: a sparql query processing baseline for big data. In *Proceedings of the 2013th International Conference on Posters & Demonstrations Track-Volume 1035*, pages 241–244. CEUR-WS. org, 2013.



- [24] W3C. Resource Description Framework (RDF). <https://www.w3.org/RDF/>, 2017. [Modificado; 14-10-2017].
- [25] W3C. SPARQL Query Language for RDF. <https://www.w3.org/TR/rdf-sparql-query/>, 2017. [Modificado; 14-10-2017].